CLONE AND FOLLOW THE SETUP INSTRUCTIONS

github.com/**CodeSequence/ngdenver2019-ngrx-workshop**

A REACTIVE STATE OF MIND

WITH ANGULAR AND NGRX

Mike Ryan

@MikeRyanDev

Mike Ryan

@MikeRyanDev

Software Architect at Synapse

Mike Ryan

@MikeRyanDev

Software Architect at Synapse

Google Developer Expert

Brandon Roberts

@brandontroberts

Brandon Roberts

@brandontroberts

Senior Angular Engineer at Nrwl

Brandon Roberts

@brandontroberts

Senior Angular Engineer at Nrwl

Google Developer Expert

Brandon Roberts

@brandontroberts

Senior Angular Engineer at Nrwl

Google Developer Expert

NgRx Core Team

Open source libraries for Angular

Open source libraries for Angular

Built with reactivity in mind

Open source libraries for Angular

Built with reactivity in mind

State management and side effects

DAY ONE SCHEDULE

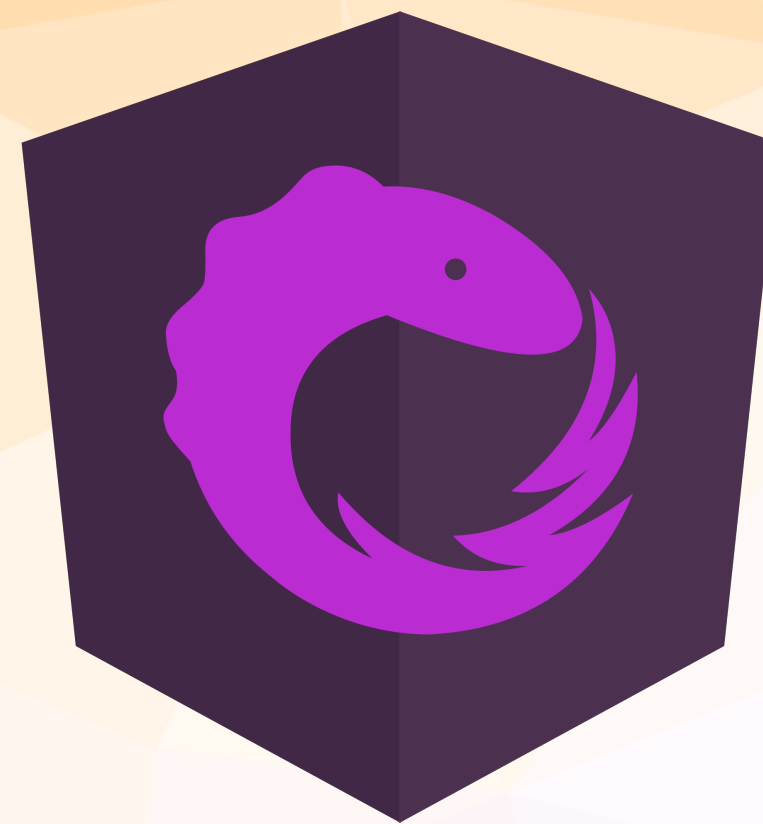- Demystifying NgRx

- Setting up the Store

- Reducers

- Actions

- Entities

- Selectors

FORMAT

1. Concept Overview

2. Demo

3. Challenge

4. Solution

The Goal

Understand the architectural implications of NgRx and how to build Angular applications with it

"How does NgRx work?"

"How does NgRx work?"

- NgRx prescribes an architecture for managing the state and side effects in you Angular application. It works by deriving a stream of updates for your application's components called the "action stream".

- You apply a pure function called a "reducer" to the action stream as a means of deriving state in a deterministic way.

- Long running processes called "effects" use RxJS operators to trigger side effects based on these updates and can optionally yield new changes back to the actions stream.
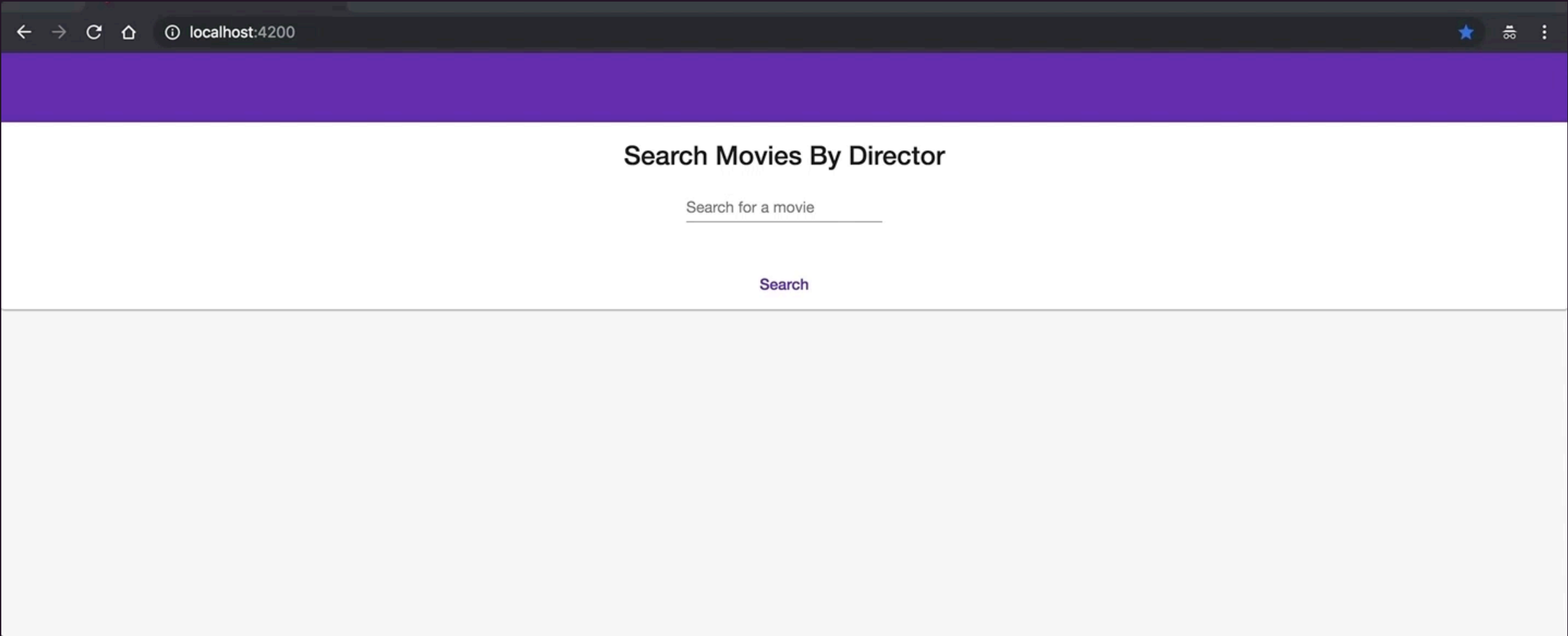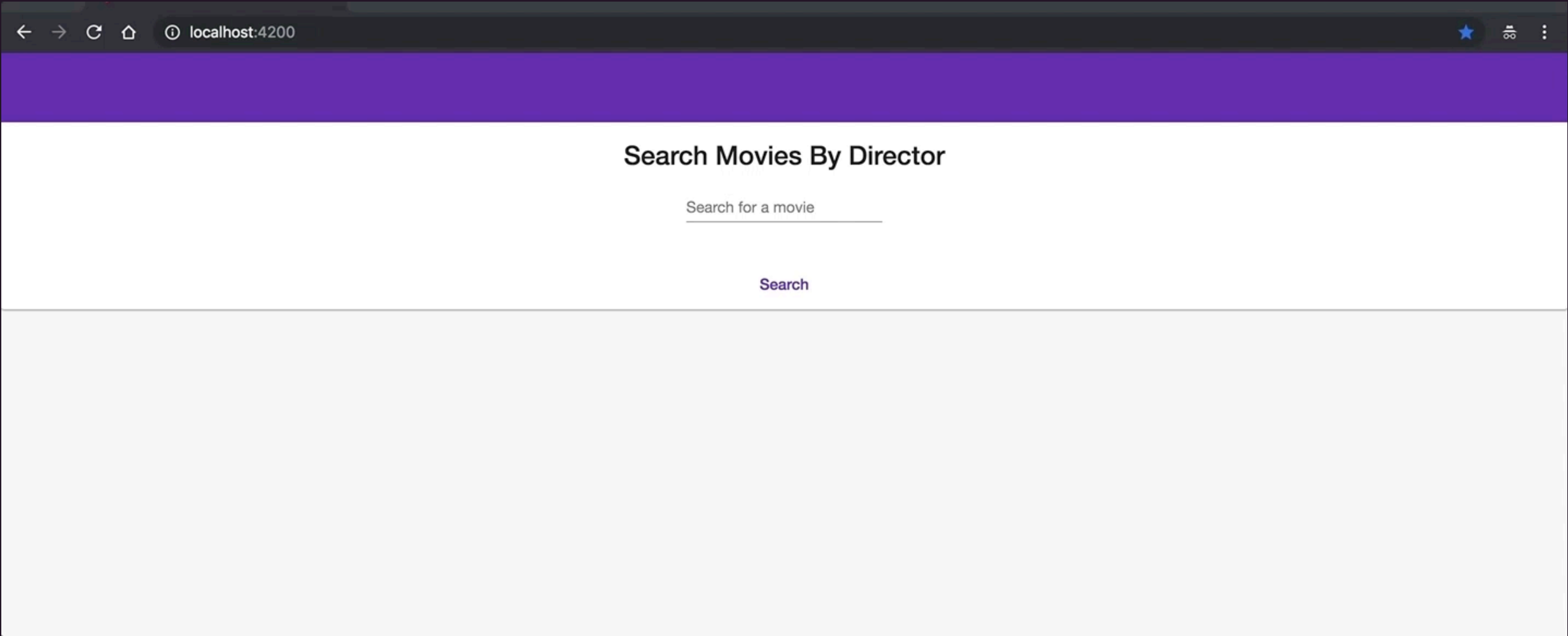
BOO THIS MAN! BOO!

BOO THIS MAN! BOO!

Let's try this a different way

You already know how NgRx works

COMPONENTS

Search Movies By Director

Search for a movie

Search

Search Movies By Director

Search for a movie

Search

Search Movies By Director

Search for a movie

Christopher Nolan

Search

Inception

Cobb, a skilled thief who commits corporate espionage by infiltrating the subconscious of his targets is offered a chance to regain his old life as payment for a task considered to be impossible: "inception", the implantation of another person's idea into a target's subconscious.

Favorite

`<movies-list-item/>`

Search Movies By Director

Search for a movie
Christopher Nolan

Search

Inception

Cobb, a skilled thief who commits corporate espionage by infiltrating the subconscious of his targets is offered a chance to regain his old life as payment for a task considered to be impossible: "inception", the implantation of another person's idea into a target's subconscious.

Favorite

`<movies-list-item/>`

`<movies-list/>`

Search Movies By Director

Search for a movie

Christopher Nolan

Search

Inception

Cobb, a skilled thief who commits corporate espionage by infiltrating the subconscious of his targets is offered a chance to regain his old life as payment for a task considered to be impossible: "inception", the implantation of another person's idea into a target's subconscious.

Favorite

`<movies-list-item/>`

`<movies-list/>`

`<search-movies-box/>`

Search Movies By Director

Search for a movie

Christopher Nolan

Search

Inception

Cobb, a skilled thief who commits corporate espionage by infiltrating the subconscious of his targets is offered a chance to regain his old life as payment for a task considered to be impossible: "inception", the implantation of another person's idea into a target's subconscious.

Favorite

`<movies-list-item/>`

`<movies-list/>`

`<search-movies-box/>`

`<search-movies-page/>`

Search Movies By Director

Search for a movie

Christopher Nolan

Search

Inception

Cobb, a skilled thief who commits corporate espionage by infiltrating the subconscious of his targets is offered a chance to regain his old life as payment for a task considered to be impossible: "inception", the implantation of another person's idea into a target's subconscious.

Favorite

```
<search-movies-page/>

<search-movies-box/>          <movies-list/>

                              <movies-list-item/>

@Input() movies: Movie[]
```

@Input() movie: Movie

```
<search-movies-page/>

<search-movies-box/>          <movies-list/>

                              <movies-list-item/>
```
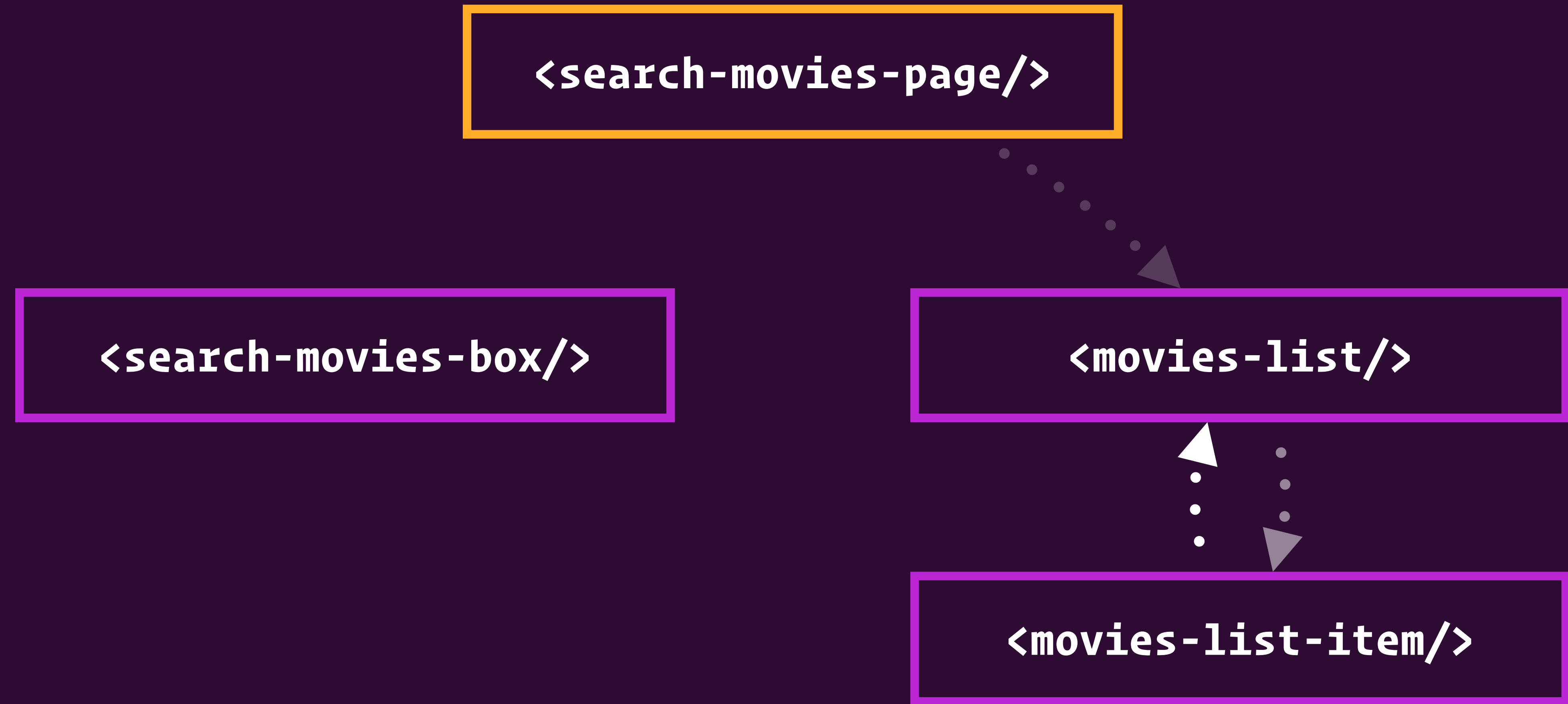
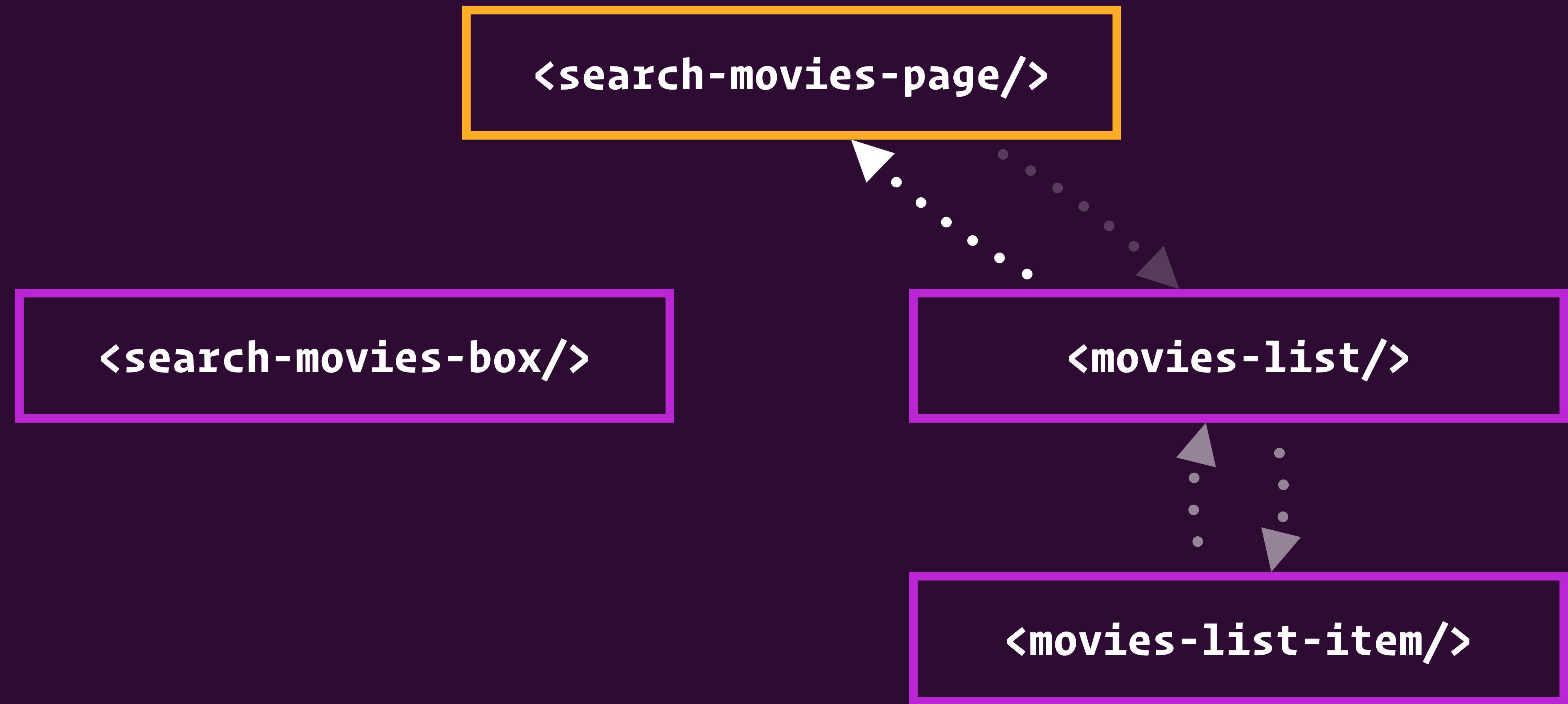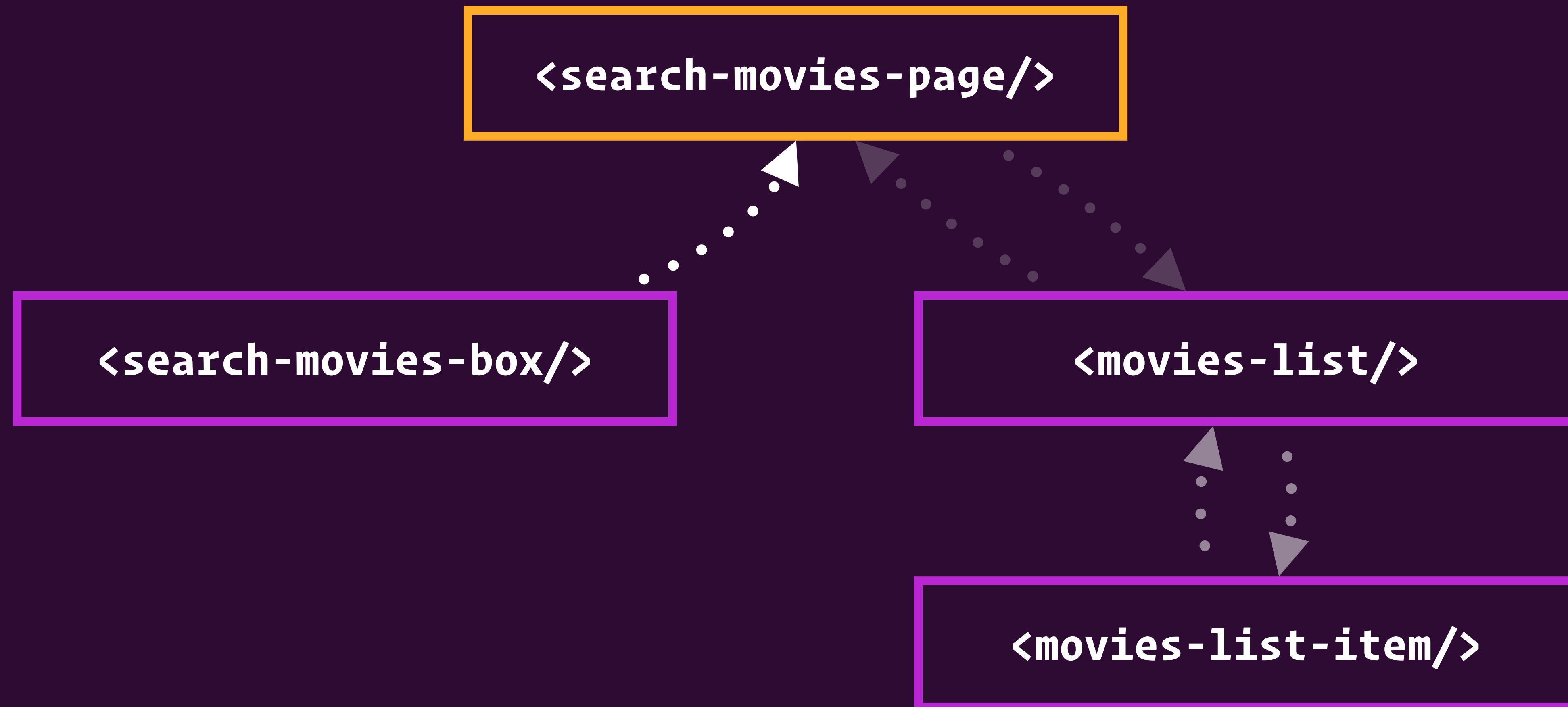@Output() favorite: EventEmitter<Movie>

```
<search-movies-page/>

<search-movies-box/>          <movies-list/>

                              <movies-list-item/>

@Output() favoriteMovie: EventEmitter<Movie>
```

```
<search-movies-page/>

<search-movies-box/>          <movies-list/>

                              <movies-list-item/>
```

@Output() search: EventEmitter<string>

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
class SearchMoviesPageComponent {
  movies: Movie[] = [];

  onSearch(searchTerm: string) {
    this.moviesService.findMovies(searchTerm)
      .subscribe(movies => {
        this.movies = movies;
      });
  }
}
```

```
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
class SearchMoviesPageComponent {
  movies: Movie[] = [];

  onSearch(searchTerm: string) {
    this.moviesService.findMovies(searchTerm)
      .subscribe(movies => {
        this.movies = movies;
      });
  }
}
```

STATE

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
class SearchMoviesPageComponent {
  movies: Movie[] = [];

  onSearch(searchTerm: string) {
    this.moviesService.findMovies(searchTerm)
      .subscribe(movies => {
        this.movies = movies;
      });
  }
}
```

```
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
class SearchMoviesPageComponent {
  movies: Movie[] = [];

  onSearch(searchTerm: string) {
    this.moviesService.findMovies(searchTerm)
      .subscribe(movies => {
        this.movies = movies;
      });
  }
}
```

SIDE EFFECT

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
class SearchMoviesPageComponent {
  movies: Movie[] = [];

  onSearch(searchTerm: string) {
    this.moviesService.findMovies(searchTerm)
      .subscribe(movies => {
        this.movies = movies;
      });
  }
}
```

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
class SearchMoviesPageComponent {
  movies: Movie[] = [];

  onSearch(searchTerm: string) {
    this.moviesService.findMovies(searchTerm)
      .subscribe(movies => {
        this.movies = movies;
      });
  }
}
```

STATE CHANGE

`<search-movies-page/>`

`<search-movies-page/>`

☑ Connects data to components

`<search-movies-page/>`

☑ Connects data to components

☑ Triggers side effects

`<search-movies-page/>`

- ✅ Connects data to components
- ✅ Triggers side effects
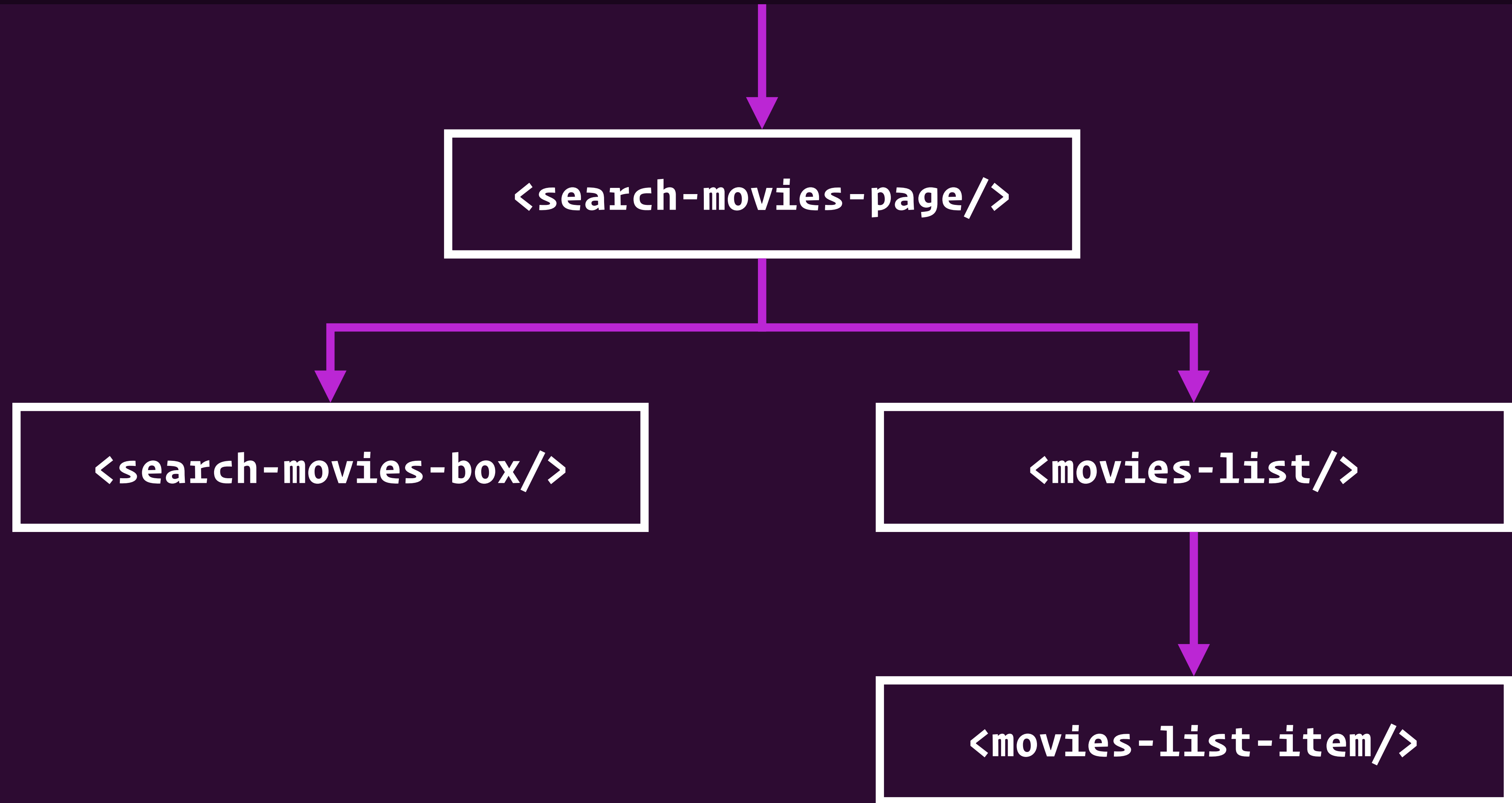- ✅ Handles state transitions

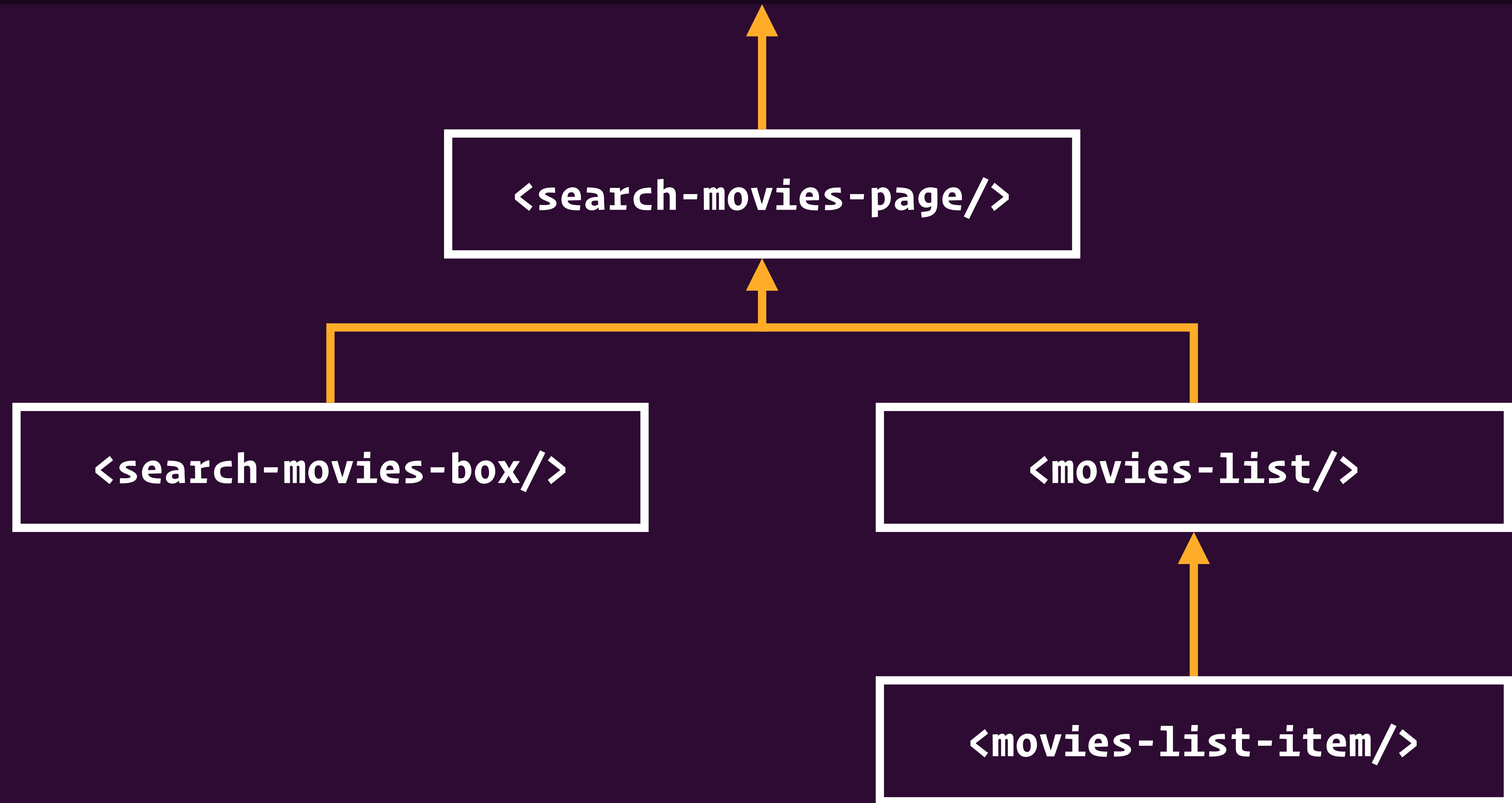# OUTSIDE WORLD

`<search-movies-page/>`

`<search-movies-box/>`

`<movies-list/>`

`<movies-list-item/>`

OUTSIDE WORLD

```
<search-movies-page/>
```

```
<search-movies-box/>
```

```
<movies-list/>
```

```
<movies-list-item/>
```

OUTSIDE WORLD

NGRX MENTAL MODEL

State flows down, changes flow up
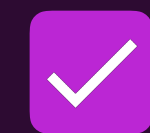
`<search-movies-page/>`

- ✅ Connects data to components
- ✅ Triggers side effects
- ✅ Handles state transitions

Single Responsibility Principle

`<search-movies-page/>`

`<search-movies-page/>`

☑ Connects data to components

@Input() and @Output()

```typescript
@Component({
  selector: 'movies-list-item',
})
export class MoviesListItemComponent {
  @Input() movie: Movie;
  @Output() favorite = new EventEmitter<Movie>();
}
```

Does this component know who is binding to its input?

```
@Component({
  selector: 'movies-list-item',
})
export class MoviesListItemComponent {
  @Input() movie: Movie;
  @Output() favorite = new EventEmitter<Movie>();
}
```

```
@Component({
  selector: 'movies-list-item',
})
export class MoviesListItemComponent {
  @Input() movie: Movie;
  @Output() favorite = new EventEmitter<Movie>();
}
```

Does this component know who is listening to its output?

```typescript
@Component({
  selector: 'movies-list-item',
})
export class MoviesListItemComponent {
  @Input() movie: Movie;
  @Output() favorite = new EventEmitter<Movie>();
}
```
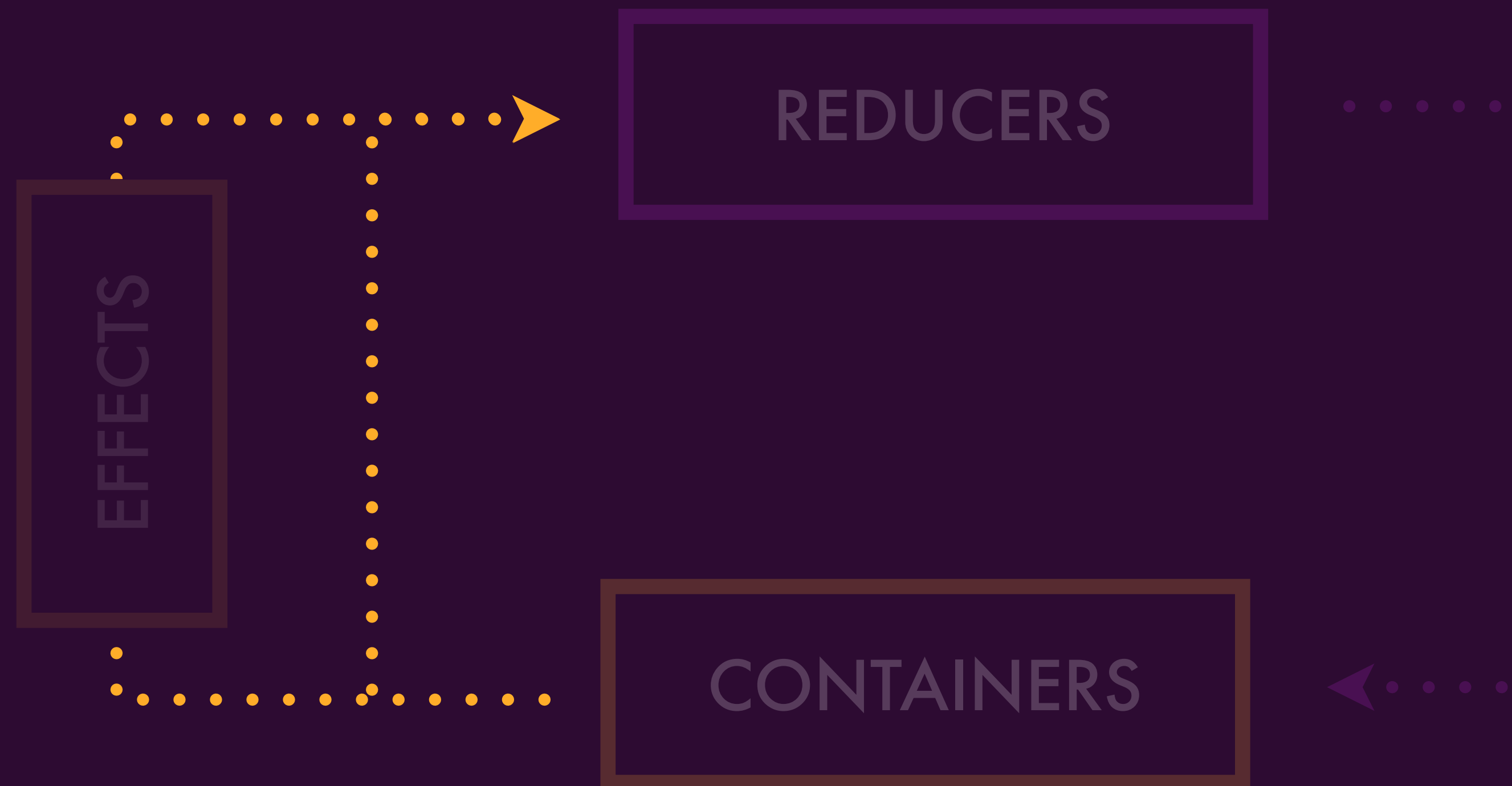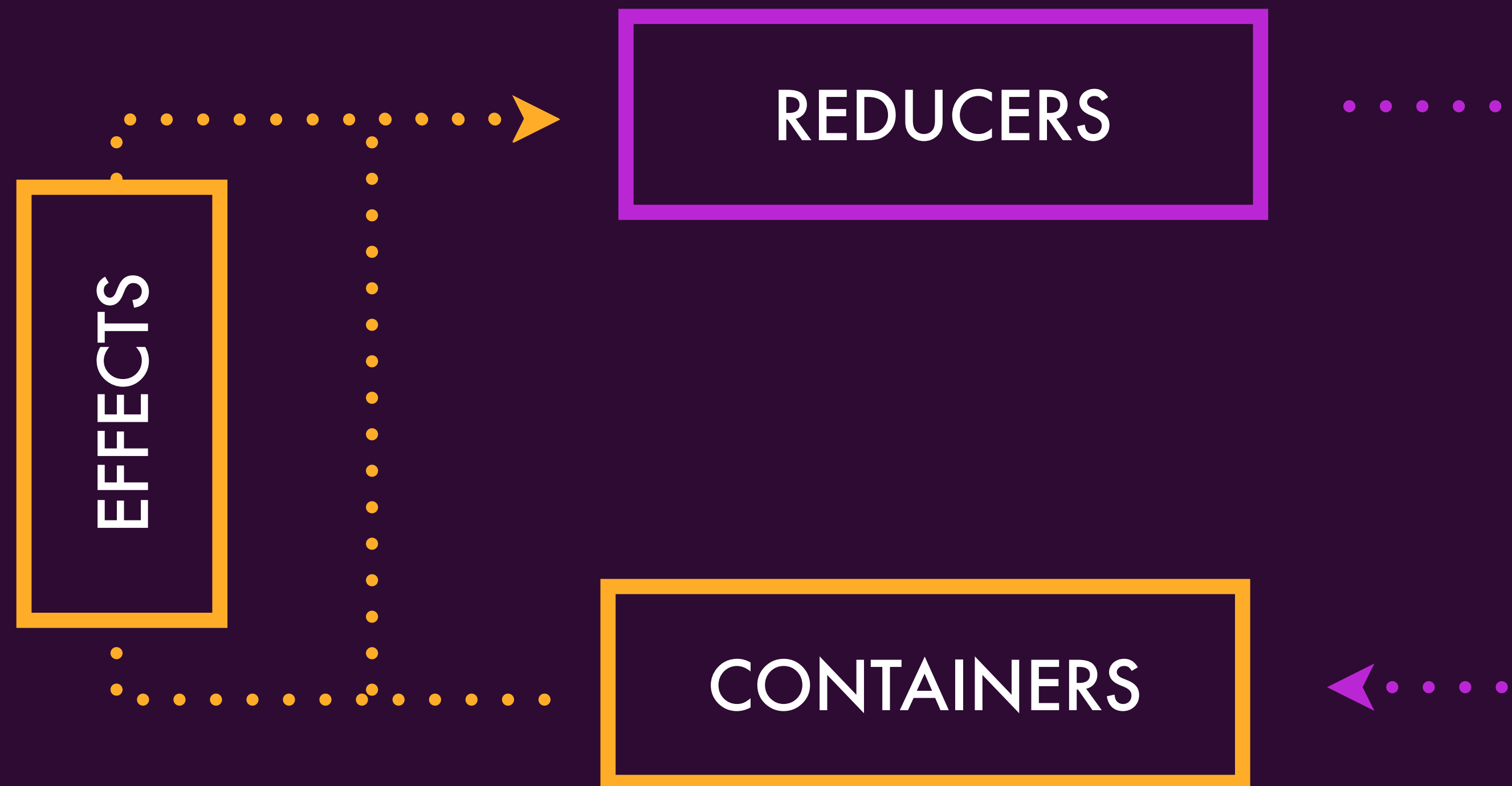
Inputs & Outputs offer Indirection

NGRX MENTAL MODEL

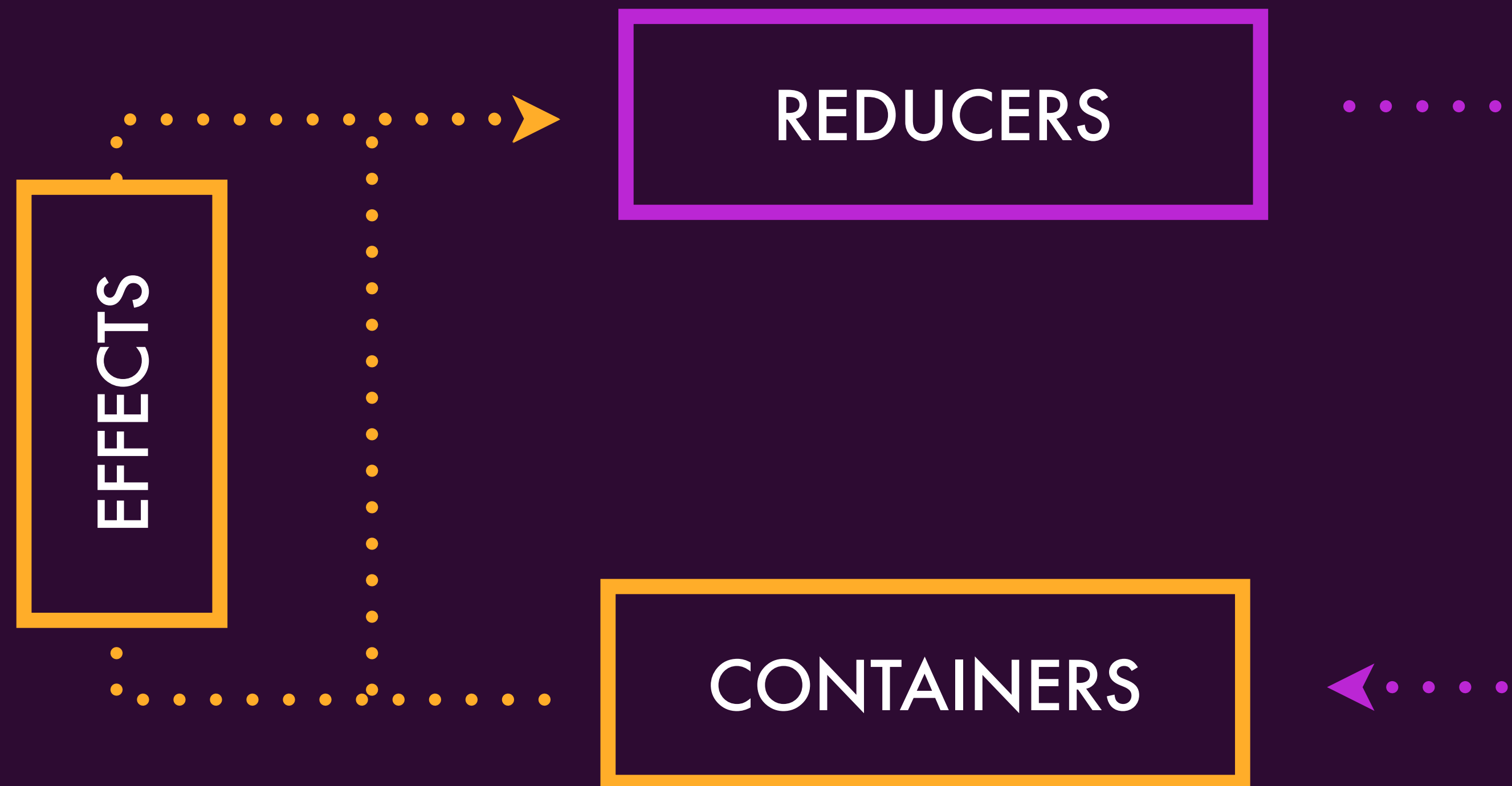*There is indirection between consumer of state, how state changes, and side effects*
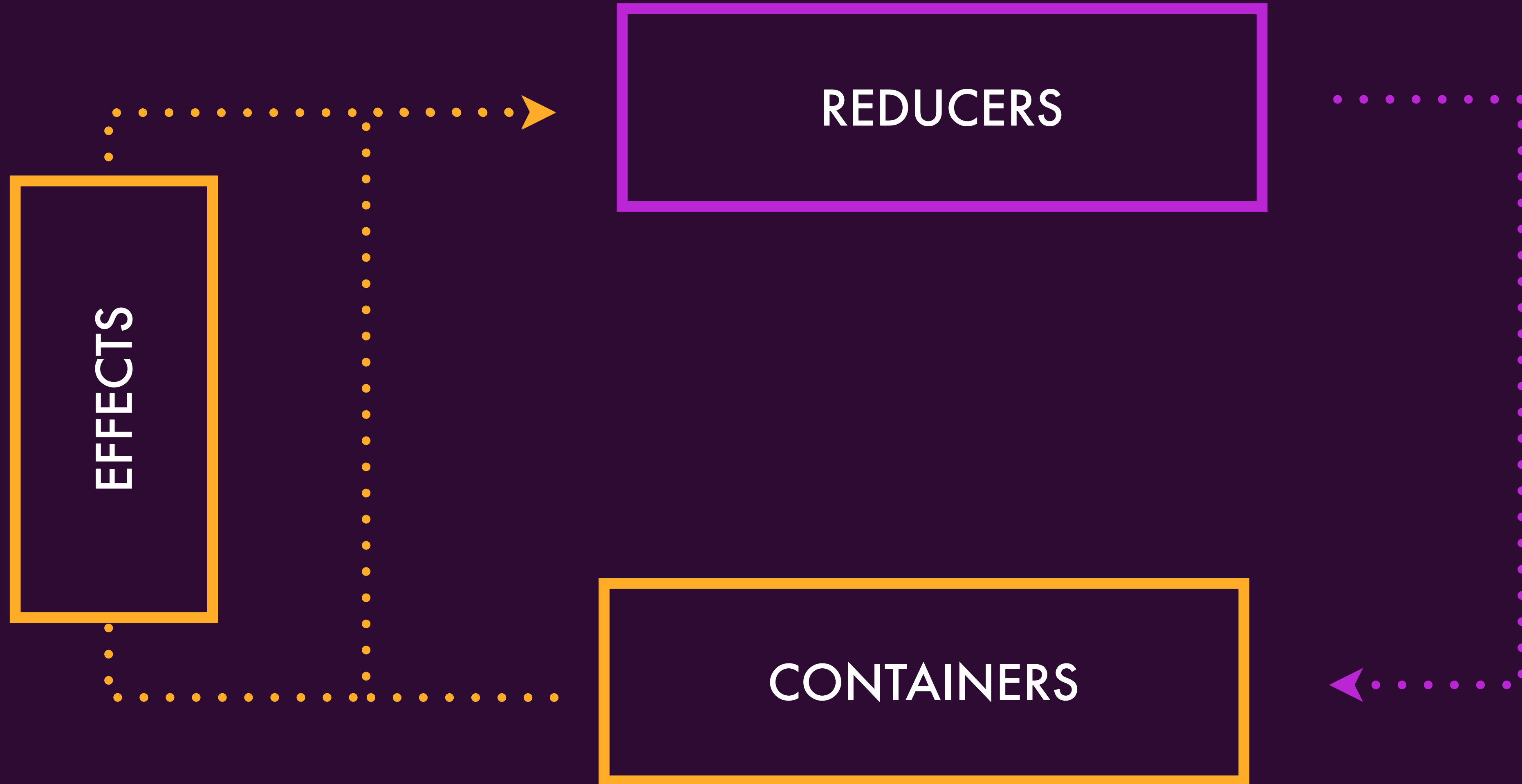
ACTIONS

```typescript
interface Action {
  type: string;
}
```
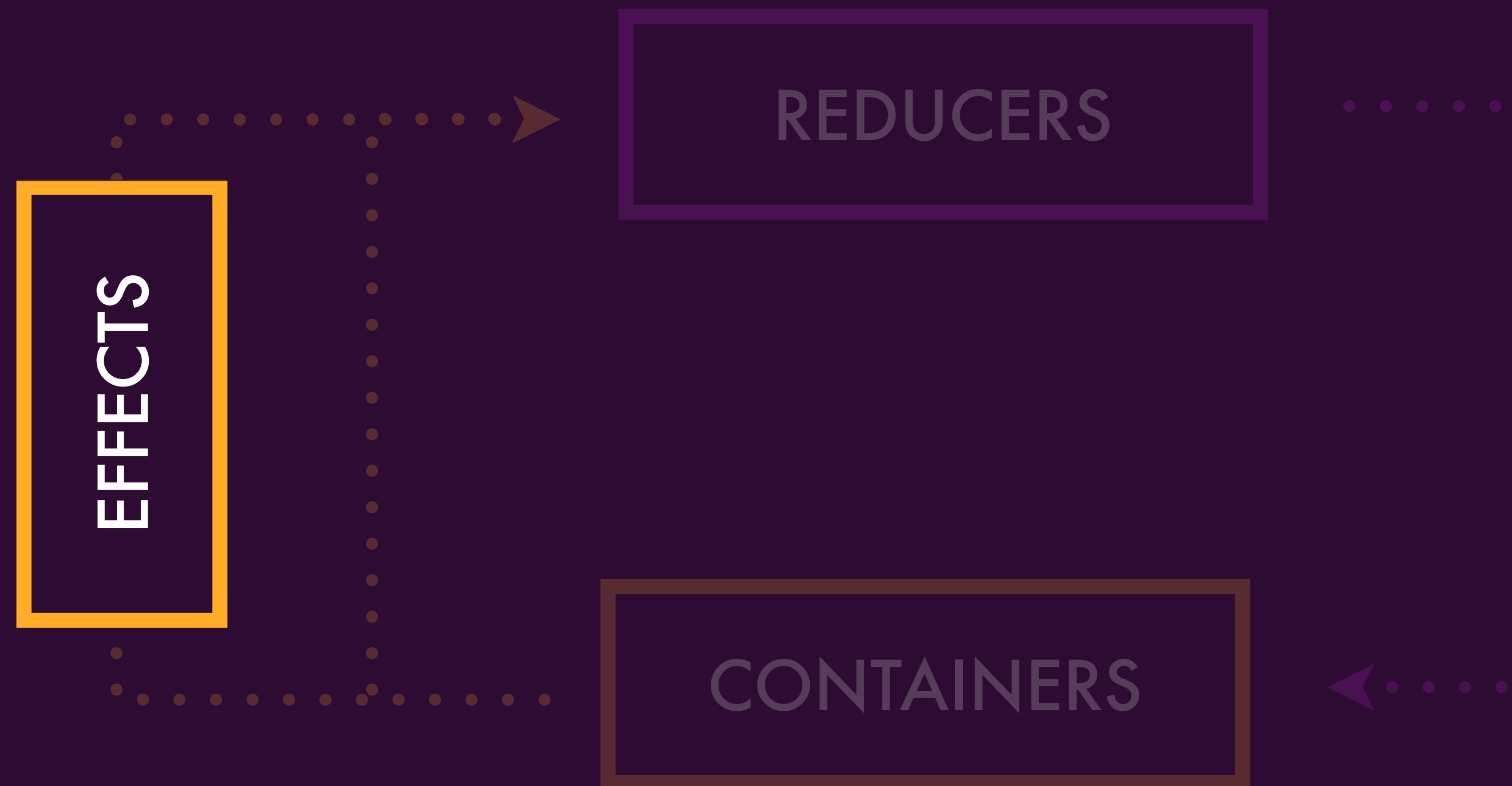
```javascript
this.store.dispatch({
  type: 'MOVIES_LOADED_SUCCESS',
  movies: [{
    id: 1,
    title: 'Enemy',
    director: 'Denis Villeneuve',
  }],
});
```

Global `@Output()` for your whole app

EFFECTS

EFFECTS

REDUCERS

CONTAINERS

EFFECTS

```
@Effect() findMovies$ = this.actions$
  .pipe(
    ofType('SEARCH_MOVIES'),
    switchMap(action => {
      return this.moviesService.findMovies(action.searchTerm)
        .pipe(
          map(movies => {
            return {
              type: 'MOVIES_LOADED_SUCCESS',
              movies,
            };
          })
        )
    }),
  );
```

```
@Effect() findMovies$ = this.actions$
  .pipe(
    ofType('SEARCH_MOVIES'),
    switchMap(action => {
      return this.moviesService.findMovies(action.searchTerm)
        .pipe(
          map(movies => {
            return {
              type: 'MOVIES_LOADED_SUCCESS',
              movies,
            };
          })
        )
    }),
  );
```

```
@Effect() findMovies$ = this.actions$
  .pipe(
    ofType('SEARCH_MOVIES'),
    switchMap(action => {
      return this.moviesService.findMovies(action.searchTerm)
        .pipe(
          map(movies => {
            return {
              type: 'MOVIES_LOADED_SUCCESS',
              movies,
            };
          })
        )
    }),
  );
```

```javascript
@Effect() findMovies$ = this.actions$
  .pipe(
    ofType('SEARCH_MOVIES'),
    switchMap(action => {
      return this.moviesService.findMovies(action.searchTerm)
        .pipe(
          map(movies => {
            return {
              type: 'MOVIES_LOADED_SUCCESS',
              movies,
            };
          })
        )
    }),
  );
```
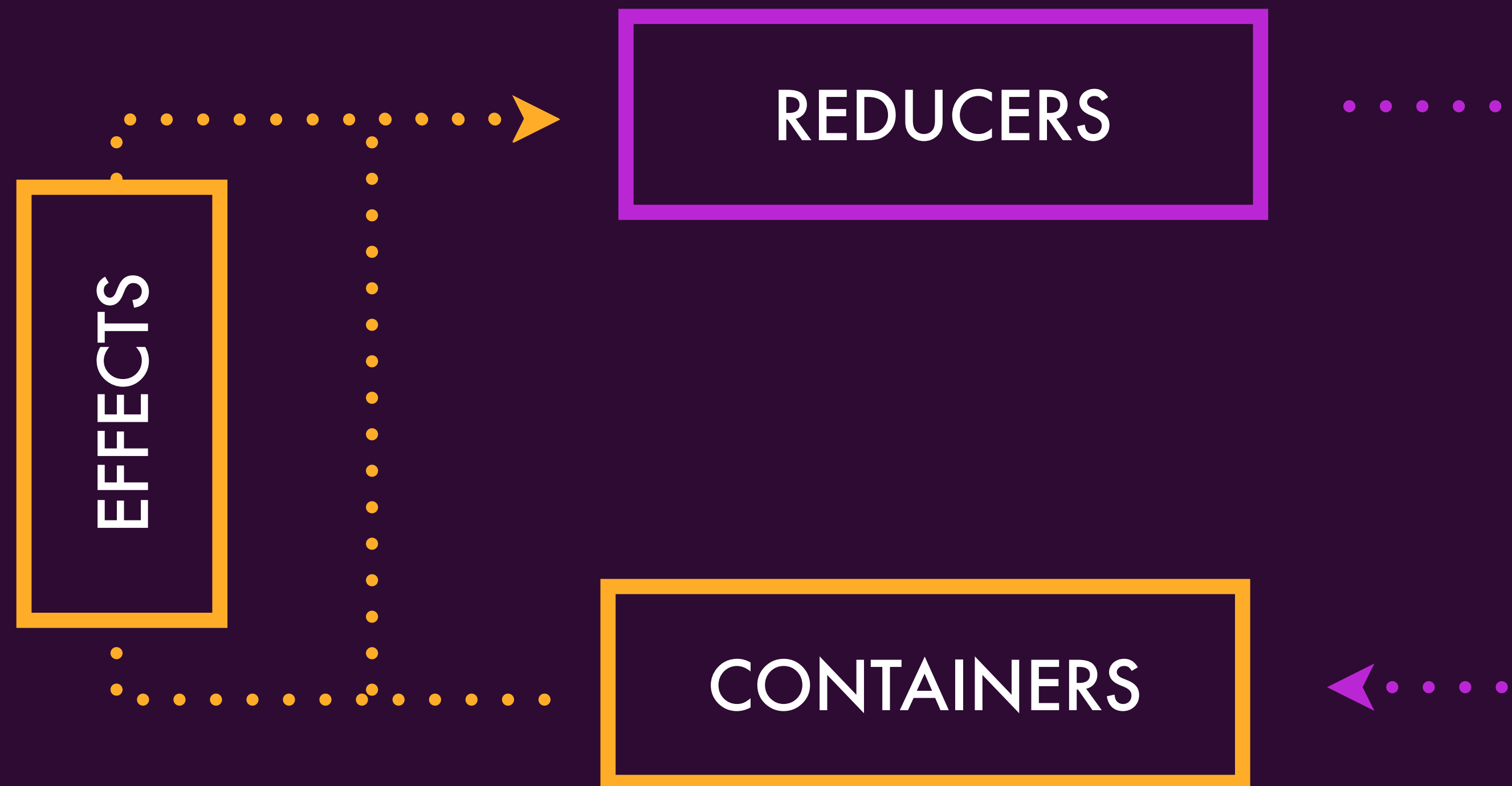
```
@Effect() findMovies$ = this.actions$
  .pipe(
    ofType('SEARCH_MOVIES'),
    switchMap(action => {
      return this.moviesService.findMovies(action.searchTerm)
        .pipe(
          map(movies => {
            return {
              type: 'MOVIES_LOADED_SUCCESS',
              movies,
            };
          })
        )
    }),
  );
```
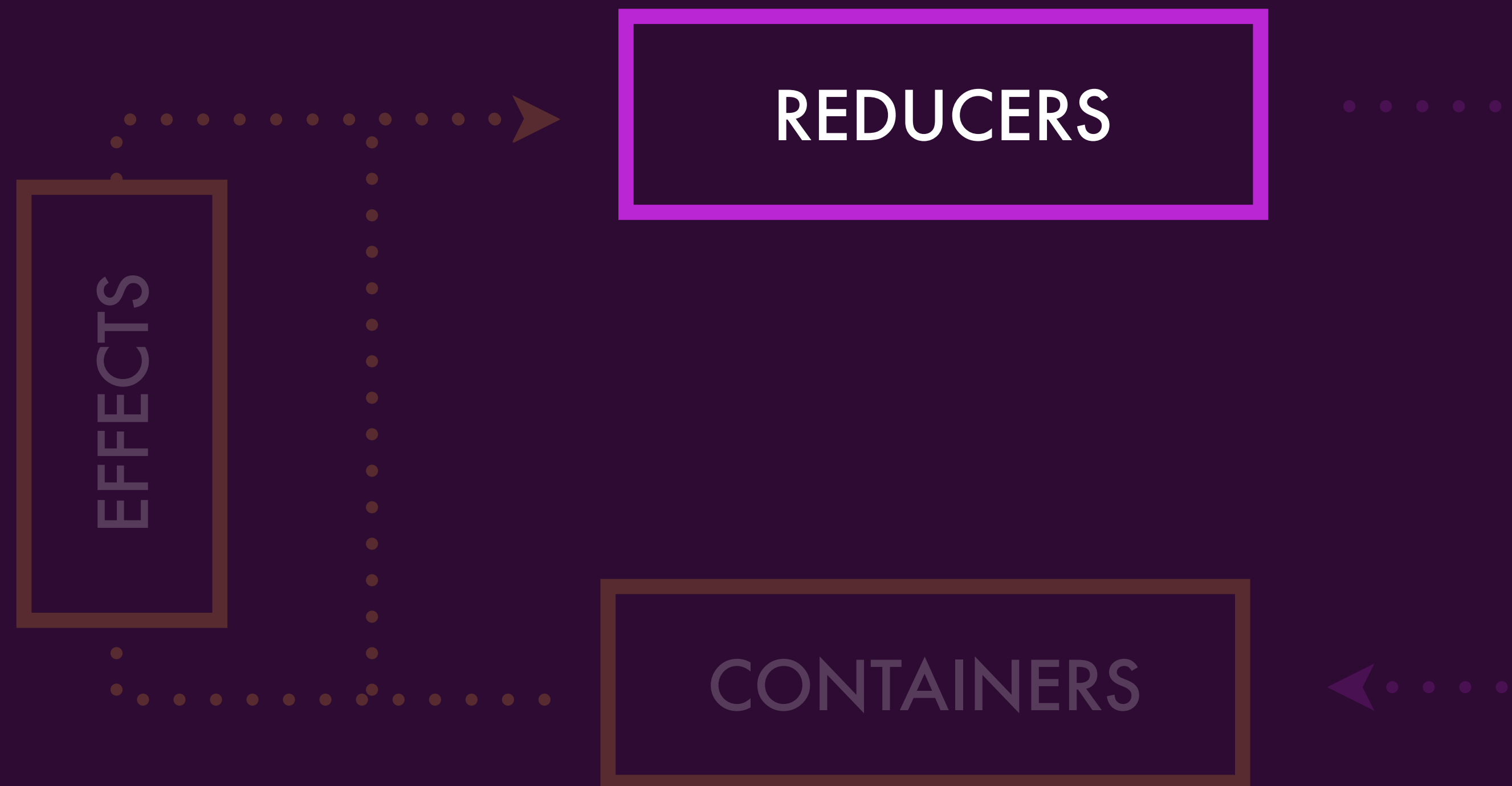
```
@Effect() findMovies$ = this.actions$
  .pipe(
    ofType('SEARCH_MOVIES'),
    switchMap(action => {
      return this.moviesService.findMovies(action.searchTerm)
        .pipe(
          map(movies => {
            return {
              type: 'MOVIES_LOADED_SUCCESS',
              movies,
            };
          })
        )
    }),
  );
```

REDUCERS

REDUCER

ACTIONS

REDUCER

ACTIONS

INITIAL STATE → REDUCER → STATE

```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }

    default: {
      return state;
    }
  }
}
```

```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }

    default: {
      return state;
    }
  }
}
```

```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }


    default: {
      return state;
    }
  }
}
```
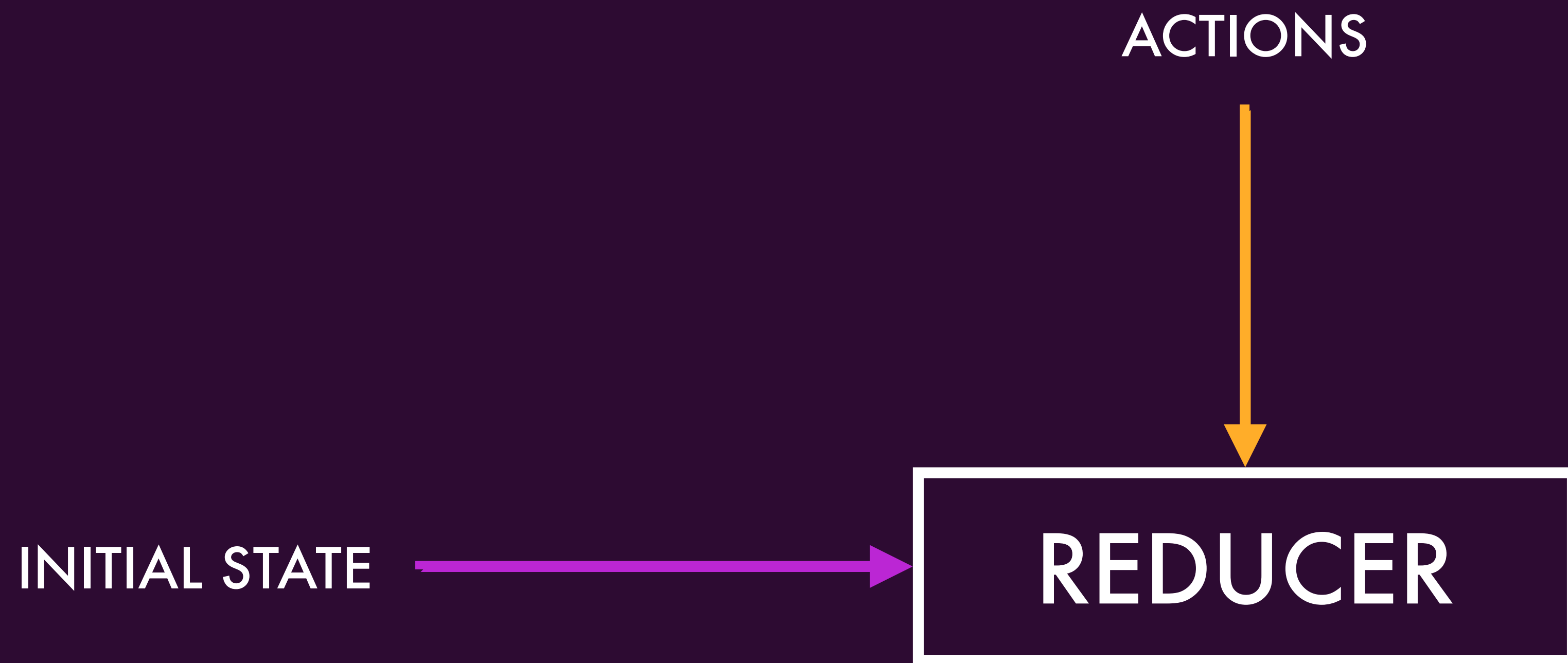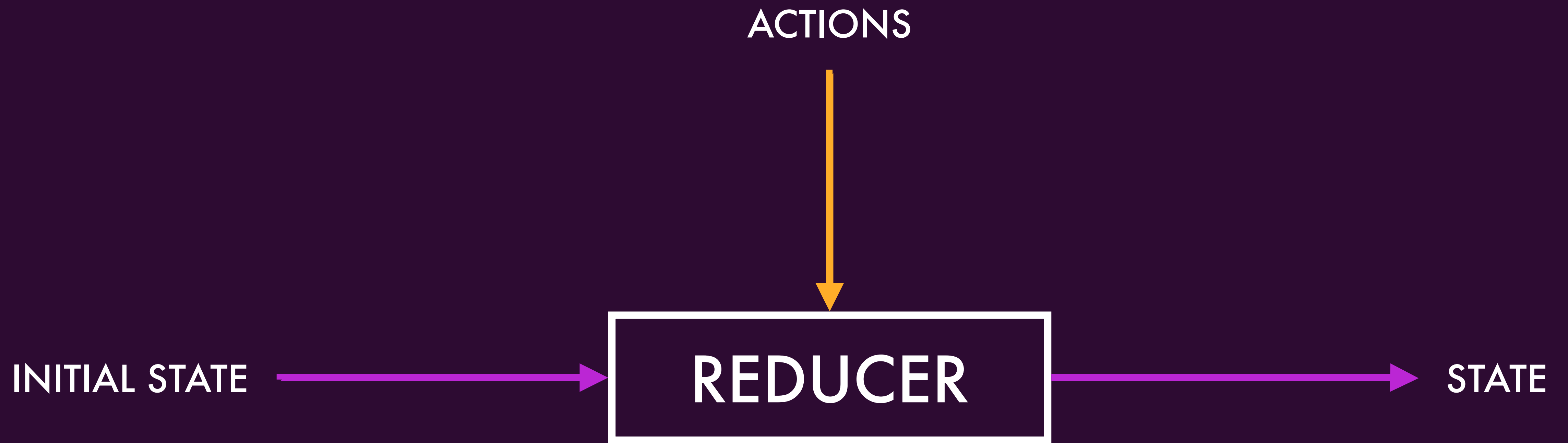
```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }

    default: {
      return state;
    }
  }
}
```
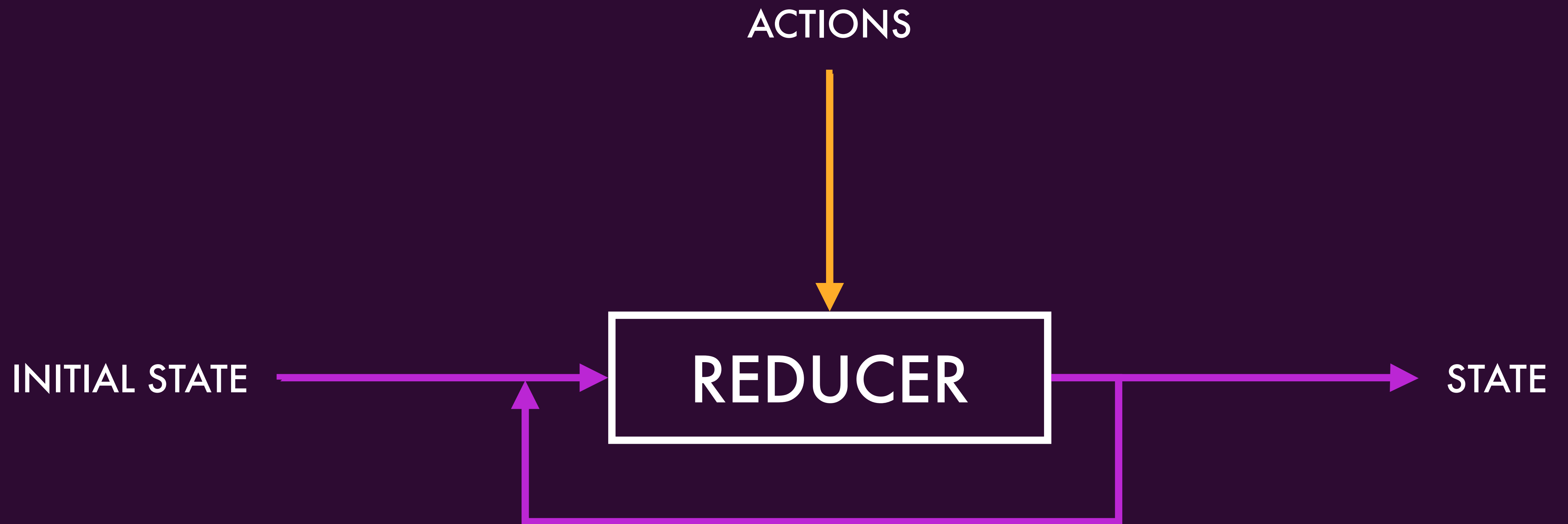
```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }

    default: {
      return state;
    }
  }
}
```

```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }

    default: {
      return state;
    }
  }
}
```

```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }

    default: {
      return state;
    }
  }
}
```

```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }

    default: {
      return state;
    }
  }
}
```

```javascript
function moviesReducer(state = [], action) {
  switch (action.type) {
    case 'MOVIES_LOADED_SUCCESS': {
      return action.movies;
    }


    default: {
      return state;
    }
  }
}
```
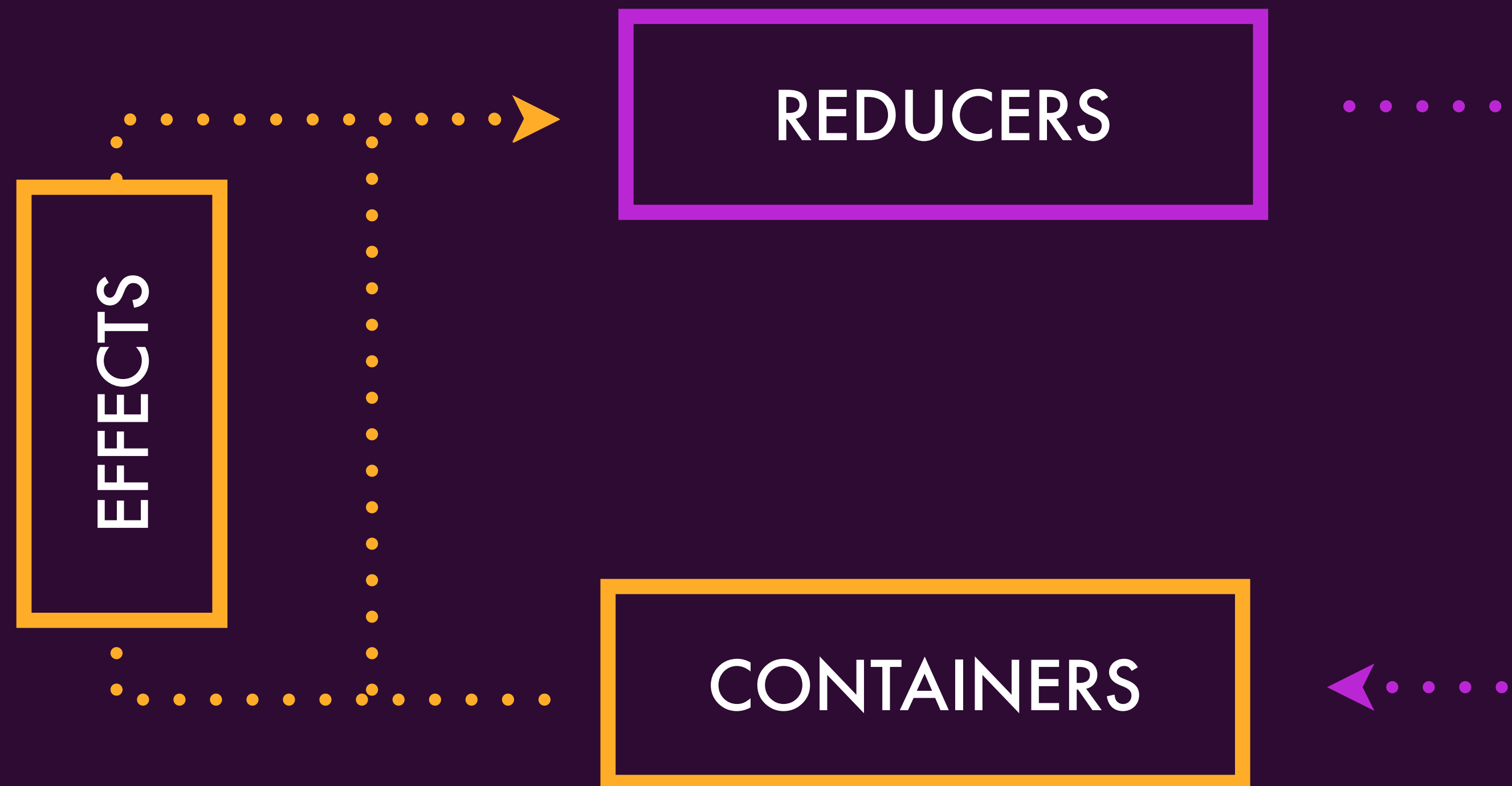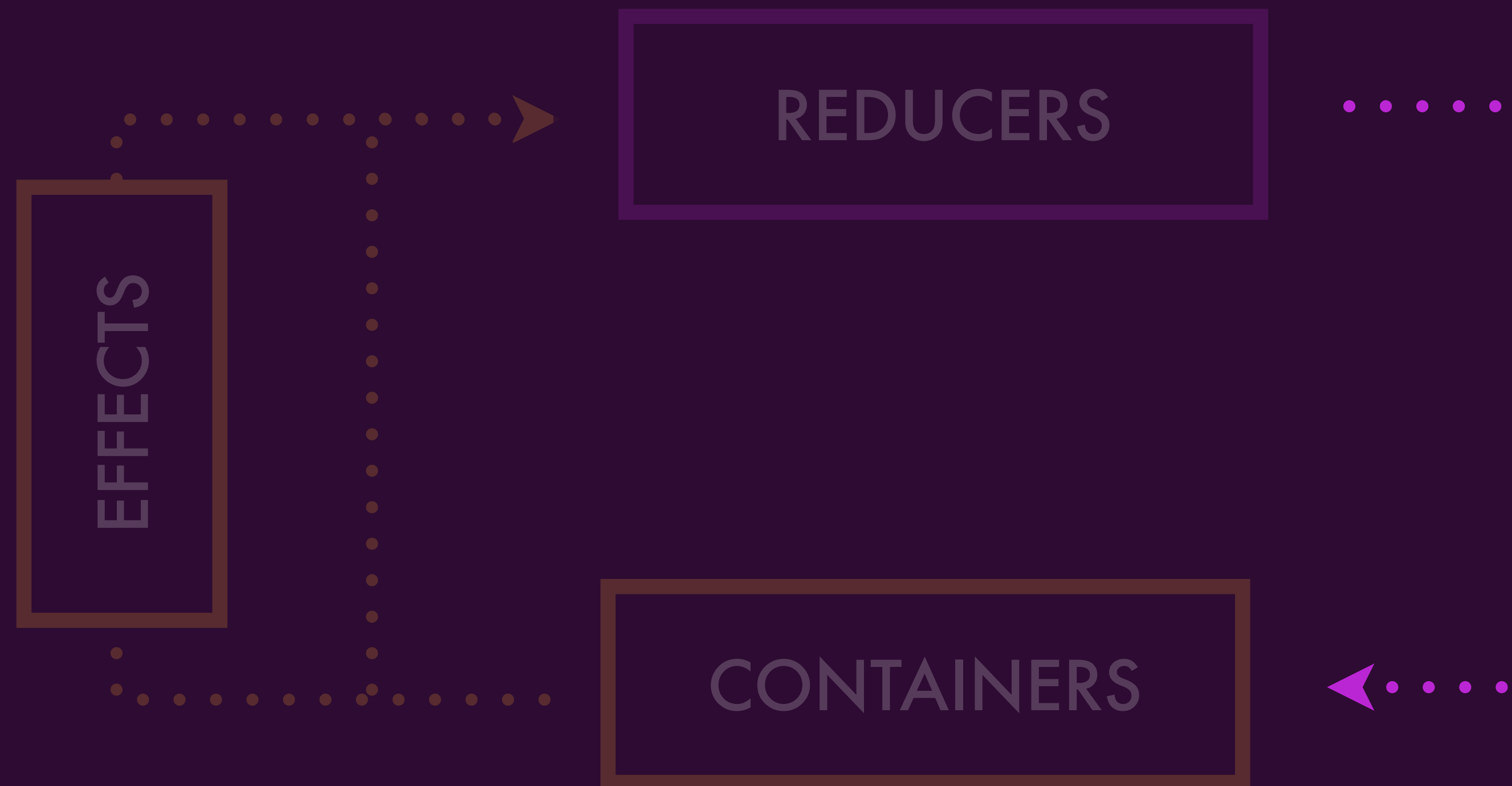
STORE .... ??? ....▶ COMPONENTS

```javascript
function selectMovies(state) {
  return state.moviesState.movies;
}
```

Global `@Input()` for your whole app

CONTAINERS

REDUCERS

EFFECTS

CONTAINERS

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies$ | async"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
export class SearchMoviesPageComponent {
  movies$: Observable<Movie[]>

  constructor(private store: Store<AppState>) {
    this.movies$ = store.select(selectMovies);
  }

  onSearch(searchTerm: string) {
    this.store.dispatch({ type: 'SEARCH_MOVIES', searchTerm });
  }
}
```

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies$ | async"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
export class SearchMoviesPageComponent {
  movies$: Observable<Movie[]>

  constructor(private store: Store<AppState>) {
    this.movies$ = store.select(selectMovies);
  }

  onSearch(searchTerm: string) {
    this.store.dispatch({ type: 'SEARCH_MOVIES', searchTerm });
  }
}
```

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies$ | async"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
export class SearchMoviesPageComponent {
  movies$: Observable<Movie[]>

  constructor(private store: Store<AppState>) {
    this.movies$ = store.select(selectMovies);
  }

  onSearch(searchTerm: string) {
    this.store.dispatch({ type: 'SEARCH_MOVIES', searchTerm });
  }
}
```

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies$ | async"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
export class SearchMoviesPageComponent {
  movies$: Observable<Movie[]>

  constructor(private store: Store<AppState>) {
    this.movies$ = store.select(selectMovies);
  }

  onSearch(searchTerm: string) {
    this.store.dispatch({ type: 'SEARCH_MOVIES', searchTerm });
  }
}
```

```typescript
@Component({
  template: `
    <search-movies-box (search)="onSearch($event)"></search-movies-box>
    <movies-list
      [movies]="movies$ | async"
      (favoriteMovie)="onFavoriteMovie($event)">
    </movies-list>
  `
})
export class SearchMoviesPageComponent {
  movies$: Observable<Movie[]>

  constructor(private store: Store<AppState>) {
    this.movies$ = store.select(selectMovies);
  }


  onSearch(searchTerm: string) {
    this.store.dispatch({ type: 'SEARCH_MOVIES', searchTerm });
  }
}
```

```
@Input() movies: Movie[]
```

```
store.select(selectMovies)
```

```typescript
@Output() search: EventEmitter<string>()
```

```typescript
this.store.dispatch({ type: 'SEARCH_MOVIES', searchTerm });
```

NGRX MENTAL MODEL

Select and Dispatch are special versions of Input and Output

RESPONSIBILITIES

RESPONSIBILITIES

☑ Containers connect data to components

RESPONSIBILITIES

- ✅ Containers connect data to components

- ✅ Effects triggers side effects

RESPONSIBILITIES

- ✅ Containers connect data to components

- ✅ Effects triggers side effects

- ✅ Reducers handle state transitions

NGRX MENTAL MODEL

Delegate responsibilities to individual modules of code

State flows down, changes flow up

✅ State flows down, changes flow up
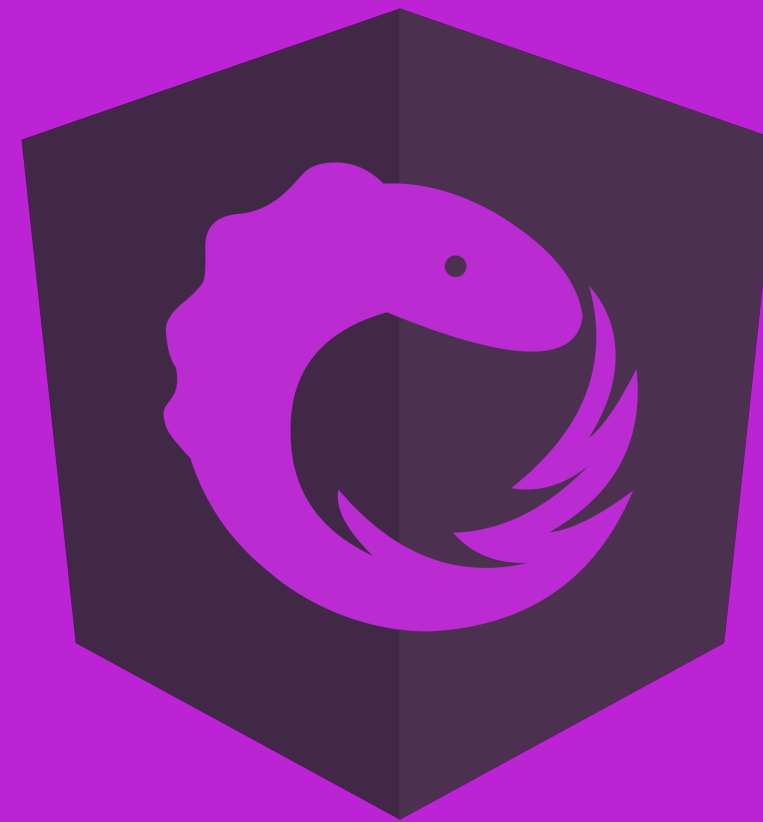
✅ Indirection between state & consumer

- ✅ State flows down, changes flow up
- ✅ Indirection between state & consumer
- ✅ Select & Dispatch => Input & Output

☑ State flows down, changes flow up

☑ Indirection between state & consumer

☑ Select & Dispatch => Input & Output

☑ Adhere to single responsibility principle

github.com/**CodeSequence/ngdenver2019-ngrx-workshop**

Demo

Challenge

1. Clone the repo at **https://github.com/CodeSequence/ngdenver2019-ngrx-workshop**
2. Checkout the **challenge** branch
3. Familiarize yourself with the **file structure**
4. Where is **movies state** handled?
5. Where are the **movies actions** located?
6. How does the **movies state** flow into the movies component?
7. How are **events** in the **movies component** going to the **movies reducer**?
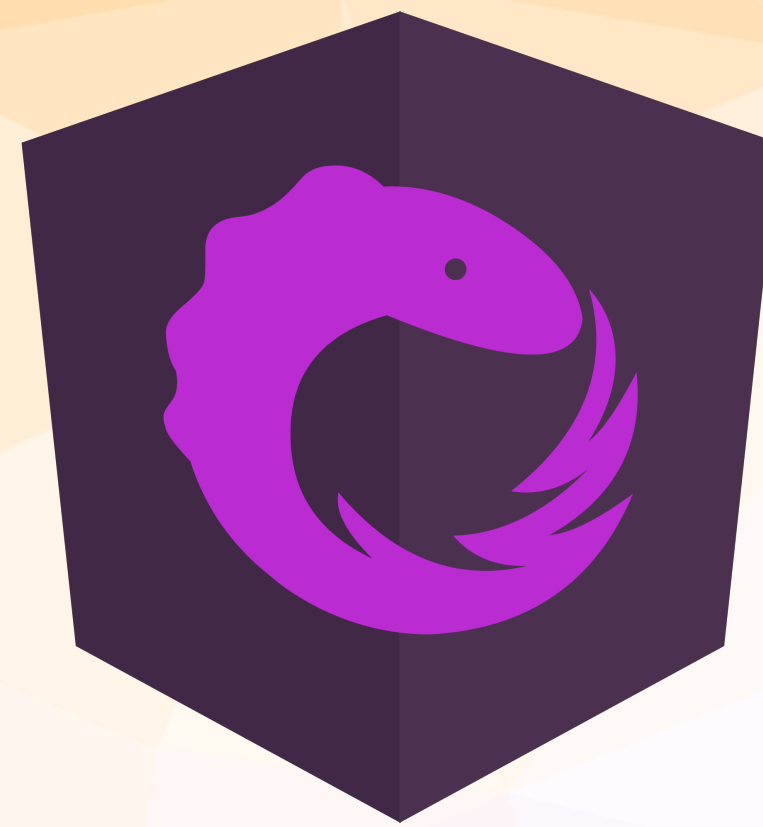
SETTING UP THE STORE

STORE

- ✅ State contained in a single state tree

- ✅ State in the store is immutable

- ✅ Slices of state are updated with reducers

```typescript
export interface MoviesState {
  activeMovieId: string | null;
  movies: Movie[];
}
```

```
export const initialState: MoviesState = {
  activeMovieId: null,
  movies: initialMovies,
};
```

```typescript
export function moviesReducer(
  state = initialState,
  action: Action
): MoviesState {
  switch (action.type) {
    default:
      return state;
  }
}
```

```typescript
import * as fromMovies from "./movies/movies.reducer";
import * as fromBooks from "./books/books.reducer";

export interface AppState {
  movies: fromMovies.MoviesState;
  books: fromBooks.BooksState;
}


export const reducers: ActionReducerMap<AppState> = {
  movies: fromMovies.reducer,
  books: fromBooks.reducer
};
```

```
@NgModule({
  imports: [
    // imports …
    StoreModule.forRoot(reducers),
    StoreDevtoolsModule.instrument({ maxAge: 5 }),
  ],
})
export class AppModule {}
```

```typescript
export class MoviesComponent implements OnInit {
  movies$: Observable<Movie[]>;
  constructor(private store: Store<AppState>) {
    this.movies$ = store.select(
      (state: AppState) => state.movies
    );
  }
}
```

```html
<app-movies-total [total]="total$ | async">
</app-movies-total>
<app-movies-list
  [movies]="movies$ | async"
  (select)="onSelect($event)"
  (delete)="onDelete($event)"
>
</app-movies-list>
```

STATE FLOWS DOWN

# Demo

Challenge, pt 1

1. Start with the **challenge** branch

2. Open **books.reducer.ts**

3. Define an interface for **BooksState** that has **activeBookId** and **books** properties

4. Define an **initialState** object that implements the **BooksState** interface

5. Create a **reducer** that defaults to **initialState** with a **default** case in a switch statement that returns **state**

Challenge, pt 2

1. Open **shared/state/index.ts** and add **books** to the **State** interface and the **books** reducer to the **reducers** object
2. Open **books-page.component.ts** and inject the **Store** service into the constructor
3. Add an **observable** property to the component that gets all of the **books** from state using the **select** operator
4. Update **books-page.component.html** to use the **async** pipe to get the list of books
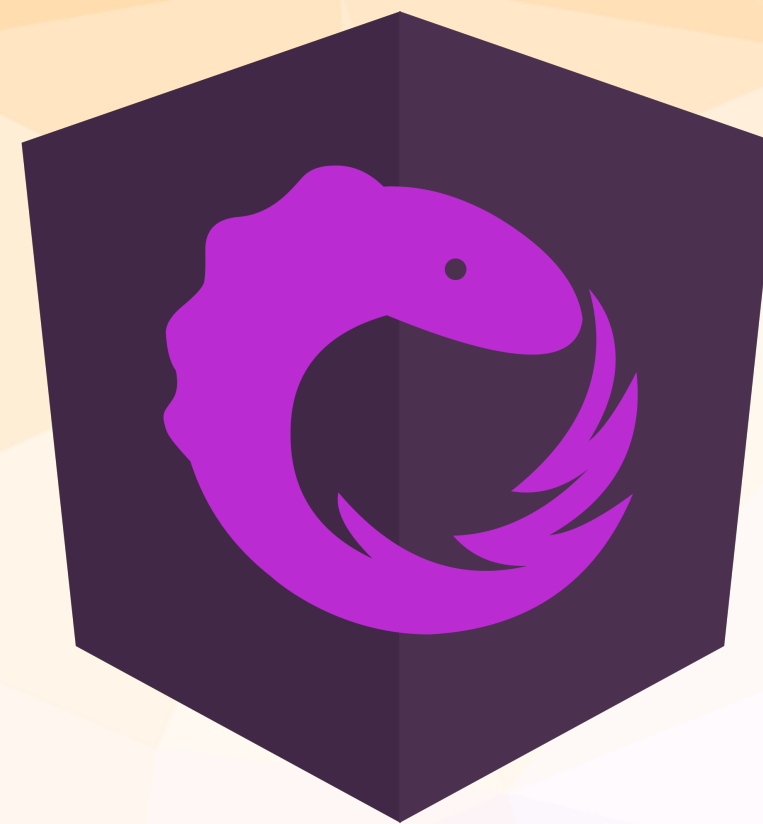
REDUCERS

REDUCERS

- ✅ Produce new states
- ✅ Receive the last state and next action
- ✅ Switch on the action type
- ✅ Use pure, immutable operations

```typescript
export function reducer(state = initialState, action: Action): MoviesState {
  switch (action.type) {
    case "select":
      return {
        activeMovieId: action.movieId,
        movies: state.movies
      };
    case "create":
      return {
        activeMovieId: state.selectedMovieId,
        movies: createMovie(state.movies, action.movie)
      };
    default:
      return state;
  }
}
```

```javascript
const createMovie = (movies, movie) => [
  ...movies,
  movie
];
const updateMovie = (movies, movie) =>
  movies.map(w => {
    return w.id === movie.id
      ? Object.assign({}, movie, w)
      : w;
  });
const deleteMovie = (movies, movie) =>
  movies.filter(w => movie.id !== w.id);
```

```
class MoviesComponent {
  createMovie(movie) {
    this.store.dispatch({
      type: "create",
      movie
    });
  }
}
```

Demo

Challenge

1. Update the books reducer to handle **"select"**, **"clear select"**, **"create"**, **"update"**, and **"delete"** actions
2. Use the helper functions already in **books.reducer.ts**
3. Update **books-page.component.ts** to dispatch **"select"**, **"clear select"**, **"create"**, **"update"**, and **"delete"** actions from the component
4. Remove the **BooksService** from the component

ACTIONS

ACTIONS

- ✅ Unified interface to describe events

- ✅ Just data, no functionality

- ✅ Has at a minimum a type property

- ✅ Strongly typed using classes and enums

GOOD ACTION HYGIENE

- ✅ Unique events get unique actions

- ✅ Actions are grouped by their source

- ✅ Actions are never reused

```
class MoviesComponent {
  createMovie(movie) {
    this.store.dispatch({
      type: "create",
      movie
    });
  }
}
```

```
export enum MoviesActionTypes {
    SelectMovie = "[Movies Page] Select Movie",
    AddMovie = "[Movies Page] Add Movie",
    UpdateMovie = "[Movies Page] Update Movie",
    DeleteMovie = "[Movies Page] Delete Movie"
}
```

```typescript
export class SelectMovie implements Action {
  readonly type = MoviesActionTypes.SelectMovie;

  constructor(public movie) {}
}
```

```typescript
export class AddMovie implements Action {
  readonly type = MoviesActionTypes.AddMovie;
  constructor(public movie: MovieModel) {}
}


export class UpdateMovie implements Action {
  readonly type = MoviesActionTypes.UpdateMovie;
  constructor(public movie: MovieModel) {}
}


export class DeleteMovie implements Action {
  readonly type = MoviesActionTypes.DeleteMovie;
  constructor(public movie: MovieModel) {}
}
```
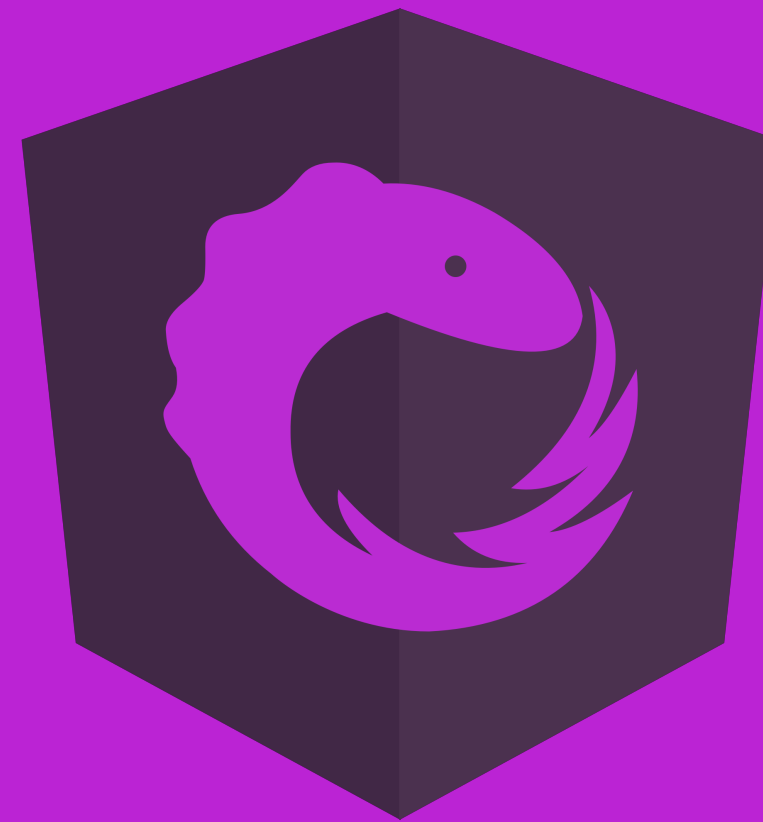
```
export type MoviesActions =
    | SelectMovie
    | AddMovie
    | UpdateMovie
    | DeleteMovie;
```

```
export function moviesReducer(
  state = initialState,
  action: MoviesActions
): MoviesState {
  switch (action.type) {
    case MoviesActionTypes.MovieSelected:
      ...
    case MoviesActionTypes.AddMovie:

      ...
    case MoviesActionTypes.UpdateMovie:

      ...
    case MoviesActionTypes.DeleteMovie:

      ...
    default:
      return state;
  }
}
```

```
createMovie(movie) {
  this.store.dispatch({
    type: "create",
    movie
  });
}
```

```
createMovie(movie) {
  this.store.dispatch(new MoviesActions.AddMovie(movie));
}
```

Demo

Challenge

1. Open **books-page.actions.ts** and create an **enum** to hold the various action types

2. Create **strongly typed actions** that adhere to **good action hygiene** for selecting a book, clearing the selection, creating a book, updating a book, and deleting a book.

3. Export the actions as a **union type**

4. Update **books-page.components.ts** and **books.reducer.ts** to use the new actions

ENTITIES

ENTITY

- ✅ Working with collections should be fast
- ✅ Collections are very common
- ✅ Common set of basic state operations
- ✅ Common set of basic state derivations

```
interface EntityState<Model> {
  ids: string[] | number[];
  entities: { [id: string | number]: Model };
}
```

```typescript
export interface MoviesState extends EntityState<Movie> {
  activeMovieId: string | null;
}
export const adapter = createEntityAdapter<Movie>();
export const initialState: Movie = adapter.getInitialState(
  {
    activeMovieId: null
  }
);
```

```typescript
export interface MoviesState extends EntityState<Movie> {
  activeMovieId: string | null;
}
export const adapter = createEntityAdapter<Movie>();
export const initialState: Movie = adapter.getInitialState(
  {

    activeMovieId: null

  }
);
```

```typescript
export interface MoviesState extends EntityState<Movie> {
  activeMovieId: string | null;
}
export const adapter = createEntityAdapter<Movie>();
export const initialState: Movie = adapter.getInitialState(
  {
    activeMovieId: null
  }
);
```

```typescript
export interface MoviesState extends EntityState<Movie> {
  activeMovieId: string | null;
}
export const adapter = createEntityAdapter<Movie>();
export const initialState: Movie = adapter.getInitialState(
  {
    activeMovieId: null
  }
);
```
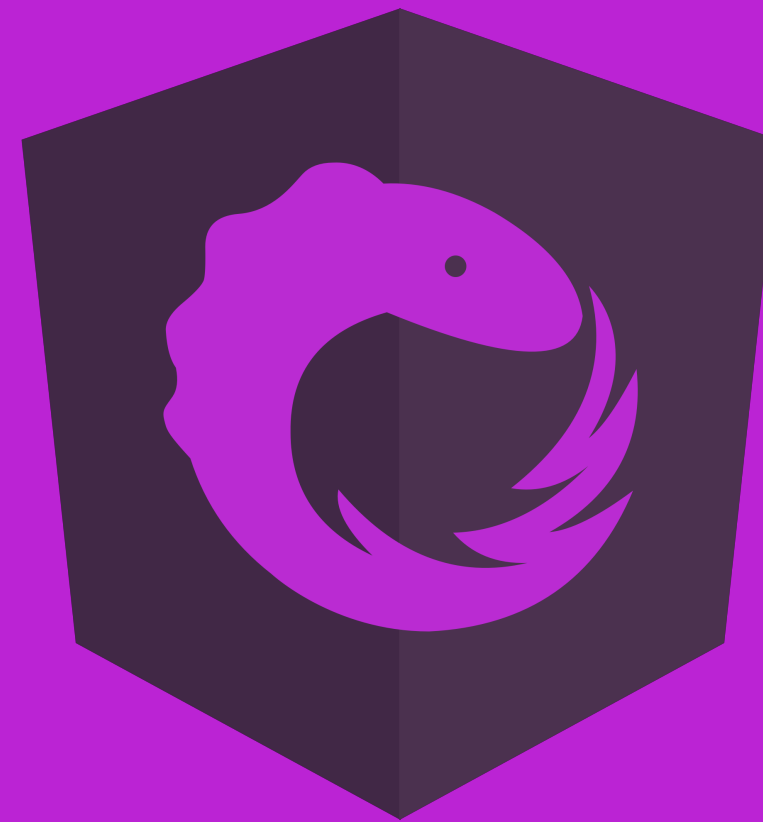
```
case MoviesActionTypes.AddMovie:
  return {
    activeMovieId: state.selectedMovieId,
    movies: createMovie(state.movies, action.movie)
  };
```

```
case MoviesActionTypes.AddMovie:
  return adapter.addOne(action.movie, state);
```

```
this.movies$ = store.select((state: any) =>
  state.movies.ids.map(id => state.movies.entities[id])
);
```

Demo

Challenge

1. Add an **"Enter"** action to **books-page.actions.ts** and dispatch it in the **getBooks()** method of **books.component.ts**

2. Update **books.reducer.ts** to use **EntityState** to define **BooksState**

3. Create an **unsorted entity adapter** for BooksState and use it to initialize initialState

4. Update the reducer to use the **adapter methods**

5. Add a case statement for the **"Enter"** action that adds all of the **initialBooks** to the state

6. Update the **books$** selector in **books-page.component.ts** to use the **ids** and **entities** properties of the **books state** to get the list of **books**
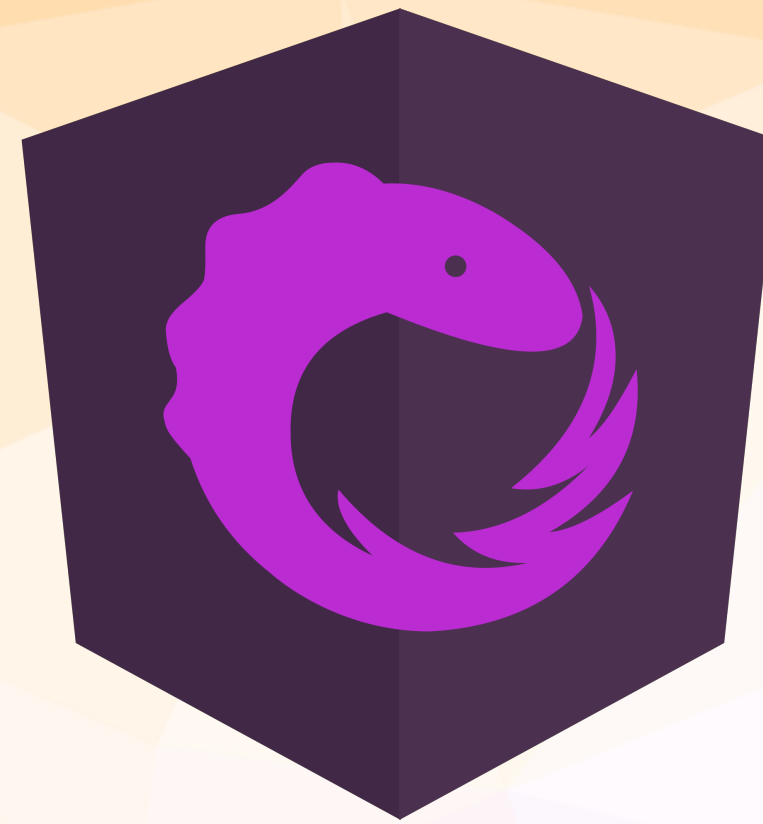
SELECTORS

SELECTORS

- ✅ Allow us to query our store for data

- ✅ Recompute when their inputs change

- ✅ Fully leverage memoization for performance

- ✅ Selectors are fully composable

```typescript
export const selectActiveMovieId = (state: MoviesState) =>
  state.activeMovieId;


// get and export the selectors
export const {
  selectIds,
  selectEntities,
  selectAll
} = adapter.getSelectors();
```

```javascript
export const selectActiveMovie = createSelector(
  selectActiveMovieId,
  selectEntities,
  (activeMovieId, movieEntities) =>
    movieEntities[activeMovieId],
);
```

```
export const selectMoviesState = (state: AppState) => state.movies;
export const selectAllMovies = createSelector(
  selectMoviesState,
  fromMovies.selectAll,
);
export const selectActiveMovie = createSelector(
  selectMoviesState,
  fromMovies.selectActiveMovie,
);
```

```
this.movies$ = store.select((state: any) =>
  state.movies.ids.map(id => state.movies.entities[id])
);
```
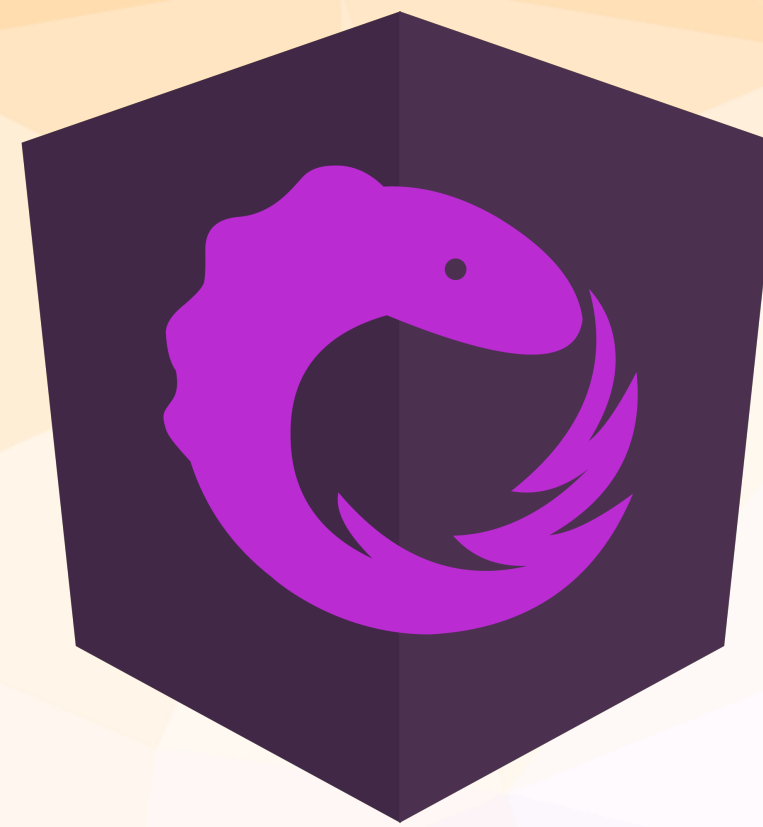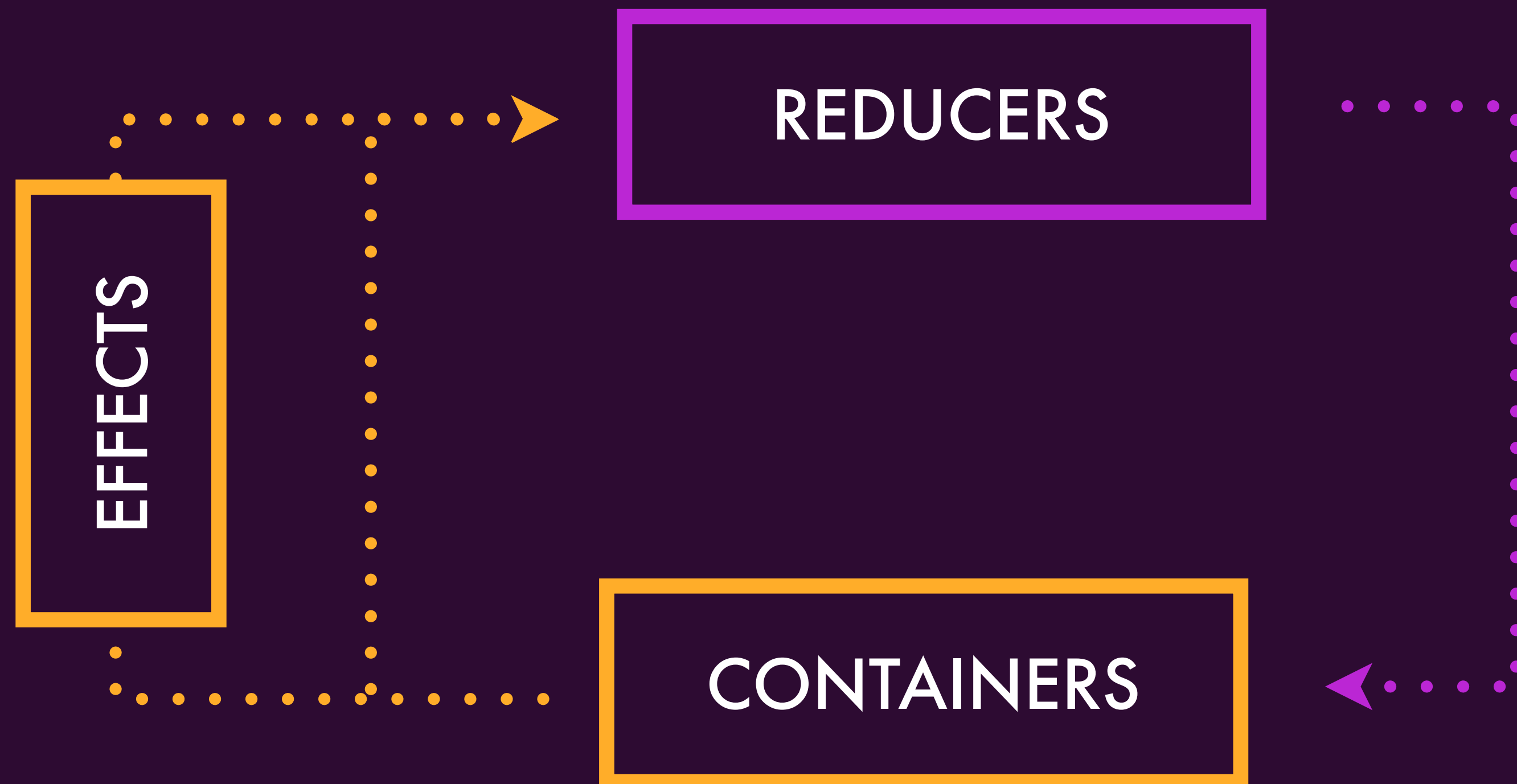
```
this.movies$ = store.select(selectAllMovies);
```
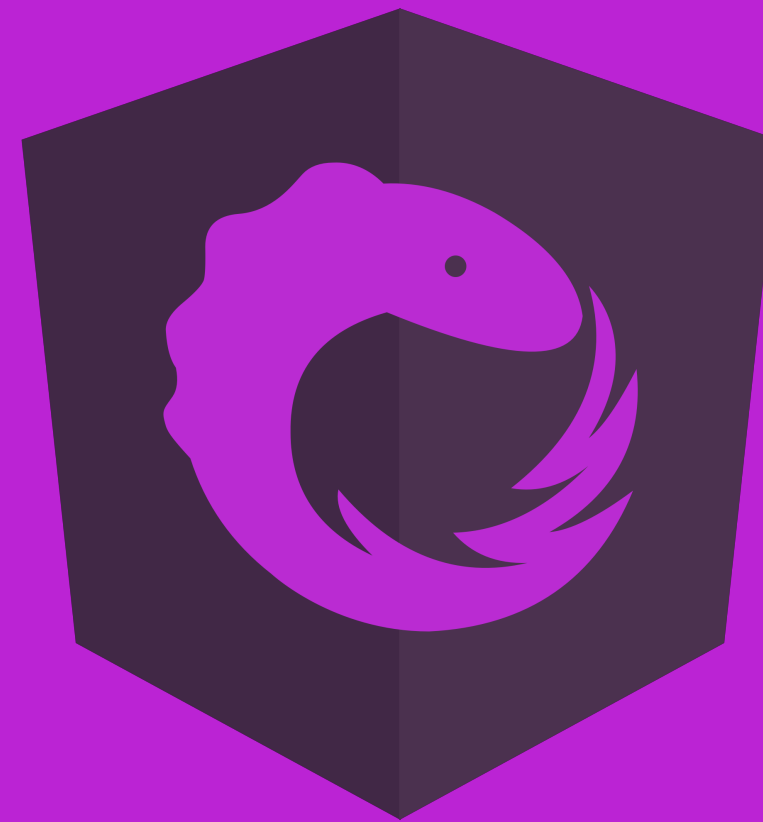
Demo

Challenge

1. Open **books.reducer.ts** and use the entity adapter to create selectors for **selectAll** and **selectEntities**

2. Write a **selector** in **books.reducer.ts** that gets **activeBookId**

3. Use **createSelector** to create a **selectActiveBook** selector using **selectEntities** and **selectActiveBookId**

4. Use **createSelector** to create a **selectEarningsTotal** selector to calculate the gross total earnings of all books using **selectAll**

5. Create global versions of **selectAllBooks, selectActiveBook,** and **selectBookEarningsTotal** in state/index.ts using **createSelector**

6. Update **books-page.component.ts** and its template to use the **selectAllBooks, selectActiveBook, and selectEarningsTotal** selectors

EFFECTS

EFFECTS

- ✅ Processes that run in the background

- ✅ Connect your app to the outside world

- ✅ Often used to talk to services

- ✅ Written entirely using RxJS streams

```typescript
export enum MoviesApiActionTypes {
  MoviesLoaded = '[Movies API] Movies Loaded',
  MovieAdded = '[Movies API] Movie Added',
  MovieUpdated = '[Movies API] Movie Updated',
  MovieDeleted = '[Movies API] Movie Deleted'
}
```

```typescript
@Injectable()
export class MoviesEffects {
  constructor(
    private actions$: Actions<
      MoviesPageActions.MoviesActions
    >,
    private moviesService: MoviesService
  ) {}
}
```

```typescript
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```typescript
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```typescript
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```typescript
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```
EffectsModule.forFeature([MoviesEffects]);
```

```typescript
const BASE_URL = "http://localhost:3000/movies";

@Injectable({ providedIn: "root" })
export class MoviesService {
  constructor(private http: HttpClient) {}

  load(id: string) {
    return this.http.get(`${BASE_URL}/${id}`);
  }
}
```

```typescript
export function moviesReducer(
  state = initialState,
  action: MoviesActions | MoviesApiActions): MoviesState {
  switch (action.type) {
    case MoviesApiActionTypes.MoviesLoaded:
      return adapter.addAll(action.movies, state);
    case MoviesApiActionTypes.MovieAdded:
      return adapter.addOne(action.movie, state);
    case MoviesActions.UpdateMovie:
      return adapter.upsertOne(action.movie, state);
    case MoviesActions.DeleteMovie:
      return adapter.removeOne(action.movie.id, state);
    default:
      return state;
  }
}
```

Demo

Challenge

1. Update **books-api.actions.ts** to export an action for **BooksLoaded** along with an **action union type**

2. Create a file at **app/books/books-api.effects.ts** and add an **effect class** to it with an **effect** called **loadBooks$** that calls **BooksService.all()** and maps the result into a **BooksLoaded** action

3. Register the **effect** using **EffectsModule.forFeature([])** in **books.module.ts**

4. Update the books **reducer** to handle the **BooksLoaded** action by replacing the **Enter** action handler

State flows down, changes flow up

✅ State flows down, changes flow up

✅ Indirection between state & consumer

- ✅ State flows down, changes flow up
- ✅ Indirection between state & consumer
- ✅ Select & Dispatch => Input & Output

- ✅ State flows down, changes flow up
- ✅ Indirection between state & consumer
- ✅ Select & Dispatch => Input & Output
- ✅ Adhere to single responsibility principle

STORE

- ✅ State contained in a single state tree
- ✅ State in the store is immutable
- ✅ Slices of state are updated with reducers

REDUCERS

- ☑ Produce new states
- ☑ Receive the last state and next action
- ☑ Switch on the action type
- ☑ Use pure, immutable operations

ACTIONS

- ☑ Unified interface to describe events

- ☑ Just data, no functionality

- ☑ Has at a minimum a type property

- ☑ Strongly typed using classes and enums

ENTITY

- ✅ Working with collections should be fast
- ✅ Collections are very common
- ✅ Common set of basic state operations
- ✅ Common set of basic state derivations

SELECTORS

- ☑ Allow us to query our store for data

- ☑ Recompute when their inputs change

- ☑ Fully leverage memoization for performance

- ☑ Selectors are fully composable

EFFECTS

- ✅ Processes that run in the background

- ✅ Connect your app to the outside world

- ✅ Often used to talk to services

- ✅ Written entirely using RxJS streams

DAY TWO SCHEDULE

- Effects++

- Advanced Actions

- Advanced Reducers

- Testing Reducers

- Testing Effects

- Q&A + Wrap Up

CLONE AND FOLLOW THE SETUP INSTRUCTIONS
github.com/**CodeSequence/ngdenver2019-ngrx-workshop**

Mike Ryan

@MikeRyanDev

Mike Ryan

@MikeRyanDev

Software Architect at Synapse

Mike Ryan

@MikeRyanDev

Software Architect at Synapse

Google Developer Expert

Mike Ryan

@MikeRyanDev

Software Architect at Synapse

Google Developer Expert

NgRx Core Team

Brandon Roberts

@brandontroberts

Brandon Roberts

@brandontroberts

Senior Angular Engineer at Nrwl

Google Developer Expert

NgRx Core Team

DAY TWO SCHEDULE

- Effects++

- Advanced Actions

- Advanced Reducers

- Testing Reducers

- Testing Effects

- Q&A + Wrap Up
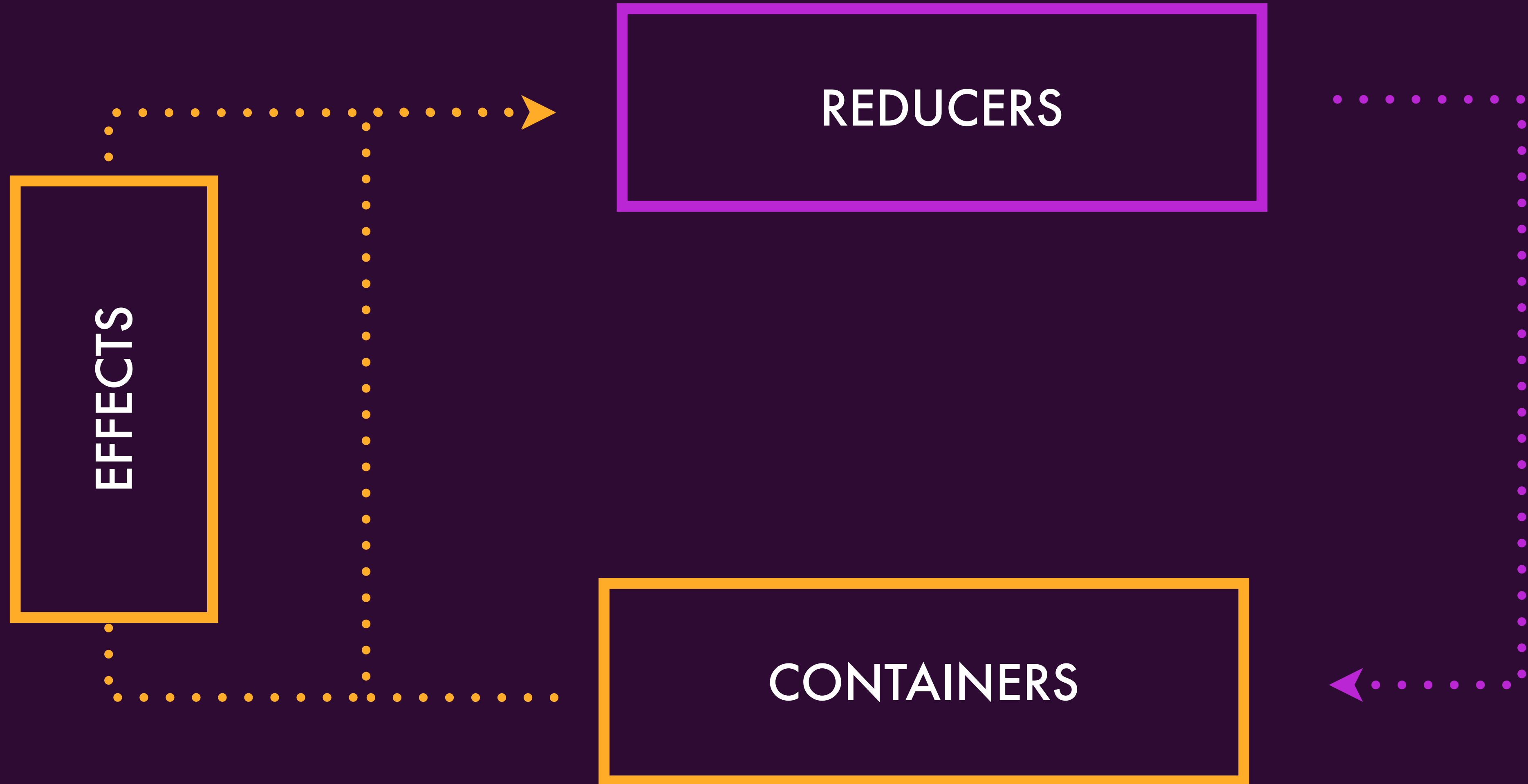
☑ State flows down, changes flow up

✅ State flows down, changes flow up

✅ Indirection between state & consumer
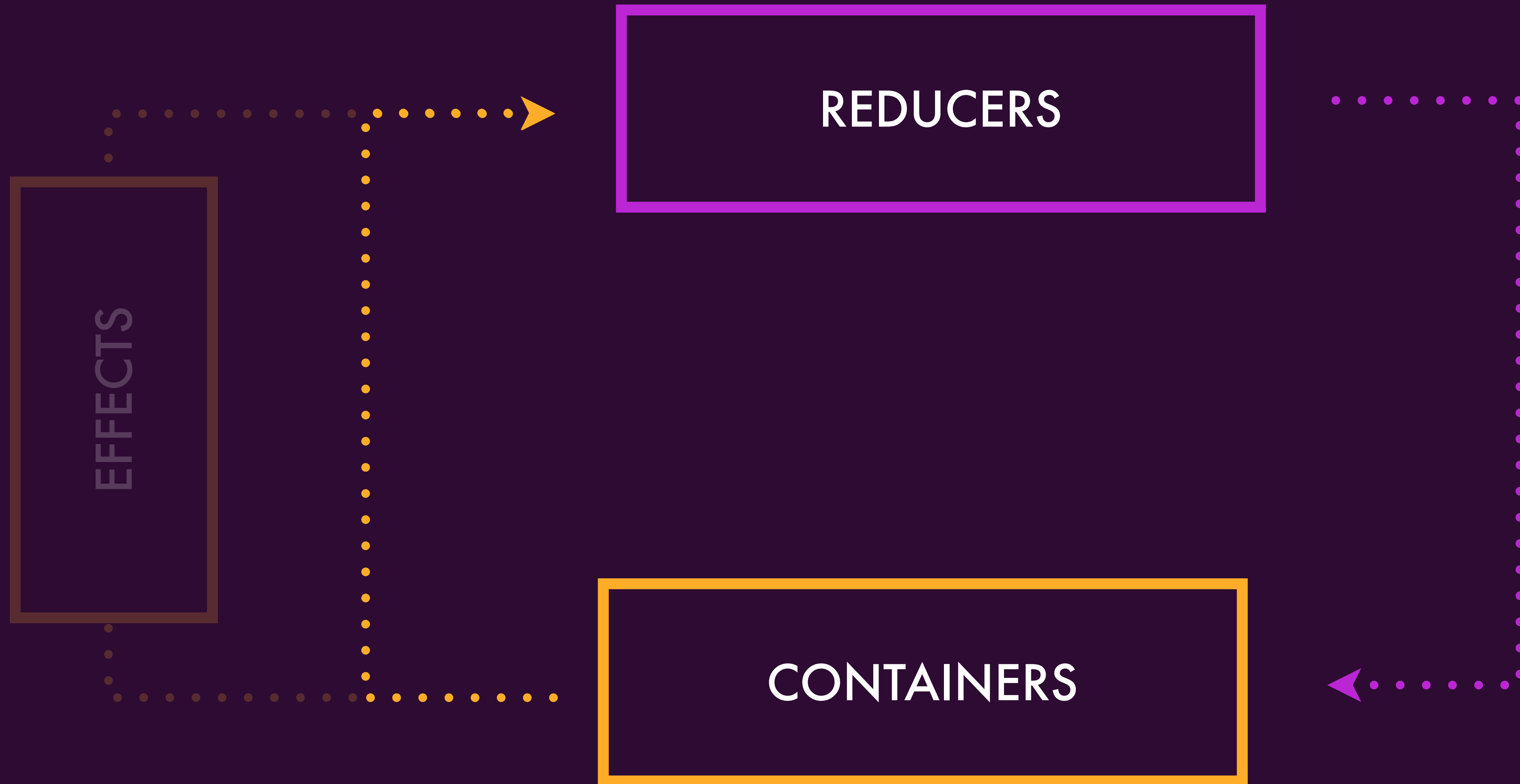
☑ State flows down, changes flow up

☑ Indirection between state & consumer

☑ Select & Dispatch => Input & Output

- ✓ State flows down, changes flow up
- ✓ Indirection between state & consumer
- ✓ Select & Dispatch => Input & Output
- ✓ Adhere to single responsibility principle

Challenge

1. Clone the repo at **https://github.com/CodeSequence/ngdenver2019-ngrx-workshop**
2. Checkout the **04-selectors** branch
3. Familiarize yourself with the **file structure**
4. Where is **books state** handled?
5. Where are the **books actions** located?
6. How does the **books state** flow into the movies component?
7. How are **events** in the **books page component** going to the **books reducer**?

EFFECTS

EFFECTS

- ✅ Processes that run in the background

- ✅ Connect your app to the outside world

- ✅ Often used to talk to services

- ✅ Written entirely using RxJS streams

```typescript
export enum MoviesApiActionTypes {
  MoviesLoaded = '[Movies API] Movies Loaded',
  MovieAdded = '[Movies API] Movie Added',
  MovieUpdated = '[Movies API] Movie Updated',
  MovieDeleted = '[Movies API] Movie Deleted'
}
```

```typescript
@Injectable()
export class MoviesEffects {
  constructor(
    private actions$: Actions<
      MoviesPageActions.MoviesActions
    >,
    private moviesService: MoviesService
  ) {}
}
```

```typescript
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```typescript
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```typescript
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```
EffectsModule.forFeature([MoviesEffects]);
```

```typescript
const BASE_URL = "http://localhost:3000/movies";

@Injectable({ providedIn: "root" })
export class MoviesService {
  constructor(private http: HttpClient) {}

  load(id: string) {
    return this.http.get(`${BASE_URL}/${id}`);
  }
}
```

```
export function moviesReducer(
  state = initialState,
  action: MoviesActions | MoviesApiActions): MoviesState {
  switch (action.type) {
    case MoviesApiActionTypes.MoviesLoaded:
      return adapter.addAll(action.movies, state);
    case MoviesApiActionTypes.MovieAdded:
      return adapter.addOne(action.movie, state);
    case MoviesActions.UpdateMovie:
      return adapter.upsertOne(action.movie, state);
    case MoviesActions.DeleteMovie:
      return adapter.removeOne(action.movie.id, state);
    default:
      return state;
  }
}
```

Demo

Challenge

1. Update **books-api.actions.ts** to export an action for **BooksLoaded** along with an **action union type**

2. Create a file at **app/books/books-api.effects.ts** and add an **effect class** to it with an **effect** called **loadBooks$** that calls **BooksService.all()** and maps the result into a **BooksLoaded** action

3. Register the **effect** using **EffectsModule.forFeature([])** in **books.module.ts**

4. Update the books **reducer** to handle the **BooksLoaded** action by replacing the **Enter** action handler

ADVANCED EFFECTS

```
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

```typescript
export class MoviesEffects {
  @Effect() loadMovies$ = this.actions$.pipe(
    ofType(MoviesActionTypes.LoadMovies),
    mergeMap(() =>
      this.moviesService.all().pipe(
        map(
          (res: MovieModel[]) =>
            new MovieApiActions.MoviesLoaded(res)
        ),
        catchError(() => EMPTY)
      )
    )
  );
}
```

WHAT MAP OPERATOR SHOULD I USE?

`mergeMap`	Subscribe immediately, never cancel or discard
`concatMap`	Subscribe after the last one finishes
`exhaustMap`	Discard until the last one finishes
`switchMap`	Cancel the last one if it has not completed

RACE CONDITIONS!

`mergeMap`	Subscribe immediately, never cancel or discard
`exhaustMap`	Discard until the last one finishes
`switchMap`	Cancel the last one if it has not completed

CONCATMAP IS THE SAFEST OPERATOR

...but there is a risk of back pressure

BACKPRESSURE DEMO

https://stackblitz.com/edit/angular-kbvxzz

`mergeMap`	Deleting items
`concatMap`	Updating or creating items
`exhaustMap`	Non-parameterized queries
`switchMap`	Parameterized queries

```
@Effect() enterMoviesPage$ = this.actions$.pipe(
  ofType(MoviesPageActions.Types.Enter),
  exhaustMap(() =>
    this.movieService.all().pipe(
      map(movies => new MovieApiActions.LoadMoviesSuccess(movies)),
      catchError(() => of(new MovieApiActions.LoadMoviesFailure()))
    )
  )
);
```

```
@Effect() updateMovie$ = this.actions$.pipe(
  ofType(MoviesPageActions.Types.UpdateMovie),
  concatMap(action =>
    this.movieService.update(action.movie.id, action.changes).pipe(
      map(movie => new MovieApiActions.UpdateMovieSuccess(movie)),
      catchError(() =>
        of(new MovieApiActions.UpdateMovieFailure(action.movie))
      )
    )
  )
);
```

Demo

Challenge

1. Add **"Book Created"**, **"Book Updated"**, and **"Book Deleted"** actions to **books-api.actions.ts** and update **books.reducer.ts** to handle these new actions
2. Open **books-api.effects.ts** and update the **loadBooks$** effect to use the **exhaustMap** operator
3. Add an effect for **creating** a book using the **BooksService.create()** method and the **concatMap** operator
4. Add an effect for **updating** a book using the **BooksService.update**() method and the **concatMap** operator
5. Add an effect for **deleting** a book using the **BooksService.delete**() method and the **mergeMap** operator

ADVANCED ACTIONS

```typescript
import { Action } from "@ngrx/store";
import { Book } from "src/app/shared/models/book.model";

export enum BooksApiActionTypes {
  BooksLoaded = "[Books API] Books Loaded Success",
  BookCreated = "[Books API] Book Created",
  BookUpdated = "[Books API] Book Updated",
  BookDeleted = "[Books API] Book Deleted"
}

export class BooksLoaded implements Action {
  readonly type = BooksApiActionTypes.BooksLoaded;

  constructor(public books: Book[]) {}
}

export class BookCreated implements Action {
  readonly type = BooksApiActionTypes.BookCreated;

  constructor(public book: Book) {}
}

export class BookUpdated implements Action {
  readonly type = BooksApiActionTypes.BookUpdated;

  constructor(public book: Book) {}
}

export class BookDeleted implements Action {
  readonly type = BooksApiActionTypes.BookDeleted;

  constructor(public book: Book) {}
}

export type BooksApiActions =
  | BooksLoaded
  | BookCreated
  | BookUpdated
  | BookDeleted;
```

```typescript
import { Action } from "@ngrx/store";
import { Book } from "src/app/shared/models/book.model";

export enum BooksApiActionTypes {
  BooksLoaded = "[Books API] Books Loaded Success",
  BookCreated = "[Books API] Book Created",
  BookUpdated = "[Books API] Book Updated",
  BookDeleted = "[Books API] Book Deleted"
}

export class BooksLoaded implements Action {
  readonly type = BooksApiActionTypes.BooksLoaded;
```

```typescript
import { Action } from "@ngrx/store";
import { Book } from "src/app/shared/models/book.model";

export enum BooksApiActionTypes {
  BooksLoaded = "[Books API] Books Loaded Success",
  BookCreated = "[Books API] Book Created",
  BookUpdated = "[Books API] Book Updated",
  BookDeleted = "[Books API] Book Deleted"
}

export class BooksLoaded implements Action {
  readonly type = BooksApiActionTypes.BooksLoaded;
```

```typescript
import { Action } from "@ngrx/store";
import { Book } from "src/app/shared/models/book.model";

export enum BooksApiActionTypes {
  BooksLoaded = "[Books API] Books Loaded Success",
  BookCreated = "[Books API] Book Created",
  BookUpdated = "[Books API] Book Updated",
  BookDeleted = "[Books API] Book Deleted"
}

export class BooksLoaded implements Action {
  readonly type = BooksApiActionTypes.BooksLoaded;
```

```typescript
export class BooksLoaded implements Action {
  readonly type = BooksApiActionTypes.BooksLoaded;

  constructor(public books: Book[]) {}
}

export class BookCreated implements Action {
  readonly type = BooksApiActionTypes.BookCreated;

  constructor(public book: Book) {}
}

export class BookUpdated implements Action {
  readonly type = BooksApiActionTypes.BookUpdated;
```

```typescript
export class BooksLoaded implements Action {
  readonly type = BooksApiActionTypes.BooksLoaded;

  constructor(public books: Book[]) {}
}

export class BookCreated implements Action {
  readonly type = BooksApiActionTypes.BookCreated;

  constructor(public book: Book) {}
}

export class BookUpdated implements Action {
  readonly type = BooksApiActionTypes.BookUpdated;
```

```typescript
export class BooksLoaded implements Action {
  readonly type = BooksApiActionTypes.BooksLoaded;

  constructor(public books: Book[]) {}
}


export class BookCreated implements Action {
  readonly type = BooksApiActionTypes.BookCreated;

  constructor(public book: Book) {}
}


export class BookUpdated implements Action {
  readonly type = BooksApiActionTypes.BookUpdated;
```

```typescript
import { Action } from "@ngrx/store";
import { Book } from "src/app/shared/models/book.model";

export enum BooksApiActionTypes {
  BooksLoaded = "[Books API] Books Loaded Success",
  BookCreated = "[Books API] Book Created",
  BookUpdated = "[Books API] Book Updated",
  BookDeleted = "[Books API] Book Deleted"
}

export class BooksLoaded implements Action {
  readonly type = BooksApiActionTypes.BooksLoaded;

  constructor(public books: Book[]) {}
}

export class BookCreated implements Action {
  readonly type = BooksApiActionTypes.BookCreated;

  constructor(public book: Book) {}
}

export class BookUpdated implements Action {
  readonly type = BooksApiActionTypes.BookUpdated;

  constructor(public book: Book) {}
}

export class BookDeleted implements Action {
  readonly type = BooksApiActionTypes.BookDeleted;

  constructor(public book: Book) {}
}

export type BooksApiActions =
  | BooksLoaded
  | BookCreated
  | BookUpdated
  | BookDeleted;
```

```typescript
import { createAction } from "@ngrx/store";
import { Book } from "src/app/shared/models/book.model";

export const booksLoaded = createAction(
  "[Books API] Books Loaded Success",
  props<{ books: Book[] }>()
);

export const bookCreated = createAction(
  "[Books API] Book Created",
  props<{ book: Book }>()
);

export const bookUpdated = createAction(
  "[Books API] Book Updated",
  props<{ book: Book }>()
);

export const bookDeleted = createAction(
  "[Books API] Book Deleted",
  props<{ book: Book }>()
);
```

```typescript
import { createAction } from "@ngrx/store";
import { Book } from "src/app/shared/models/book.model";

export const booksLoaded = createAction(
  "[Books API] Books Loaded Success",
  props<{ books: Book[] }>()
);

export const bookCreated = createAction(
  "[Books API] Book Created",
  props<{ book: Book }>()
);

export const bookUpdated = createAction(
  "[Books API] Book Updated",
  props<{ book: Book }>()
);

export const bookDeleted = createAction(
  "[Books API] Book Deleted",
  props<{ book: Book }>()
);
```

```
export const loadMoviesFailure = createAction(
  "[Movies API] Load Movies Failure"
);


MovieApiActions.loadMoviesFailure();
```

```
export const updateMovieSuccess = createAction(
  "[Movies API] Update Movie Success",
  props<{ movie: Movie }>()
);


MovieApiActions.updateMoviesSuccess({ movie });
```

```typescript
export type Union = ReturnType<
  | typeof loadMoviesSuccess
  | typeof loadMoviesFailure
  // ...
>;
```

```
@Effect() enterMoviesPage$ = this.actions$.pipe(
  ofType(MoviesPageActions.enter),
  exhaustMap(() =>
    this.movieService.all().pipe(
      map(movies => MovieApiActions.loadMoviesSuccess({ movies })),
      catchError(() => of(MovieApiActions.loadMoviesFailure()))
    )
  )
);
```
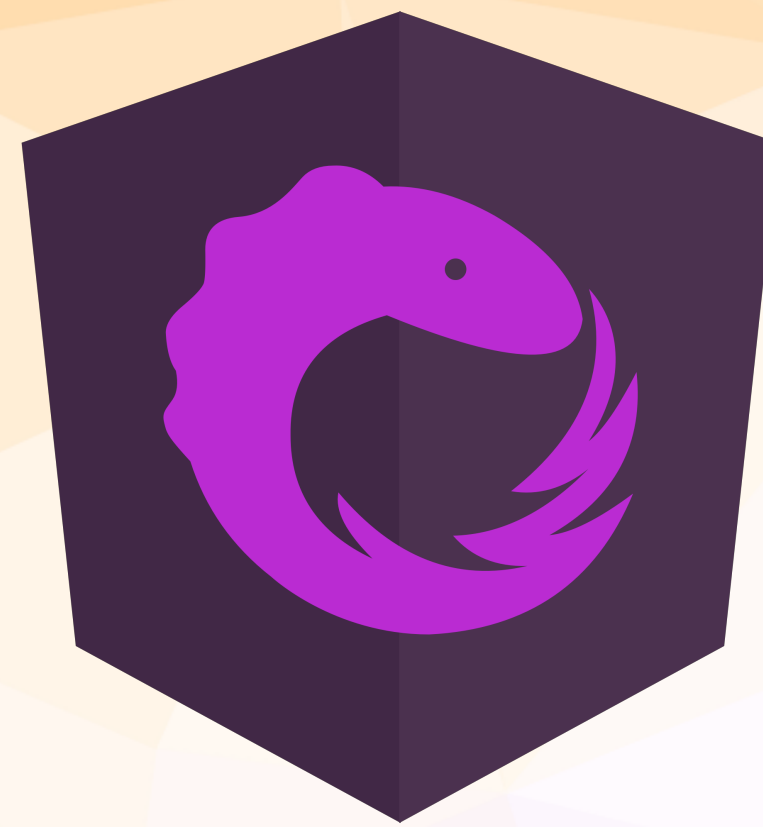
```
export function reducer(
  state: State = initialState,
  action: MovieApiActions.Union | MoviesPageActions.Union
): State {
  switch (action.type) {
    case MoviesPageActions.enter.type: {
      return { ...state, activeMovieId: null };
    }


    default: {
      return state;
    }
  }
}
```

Demo

Challenge

1. Update **books-page.actions.ts** to use the **createAction** helper and the **props factory function**

2. Use the **ReturnType** utility type to replace the action union

3. Update **books-api.actions.ts** to use the **createAction** helper and the **props factory function**

4. Use the **ReturnType** utility type to replace the action union

5. Update **books-page.component.ts, books-api.effects.ts,** and **books.reducer.ts** to use the new action format
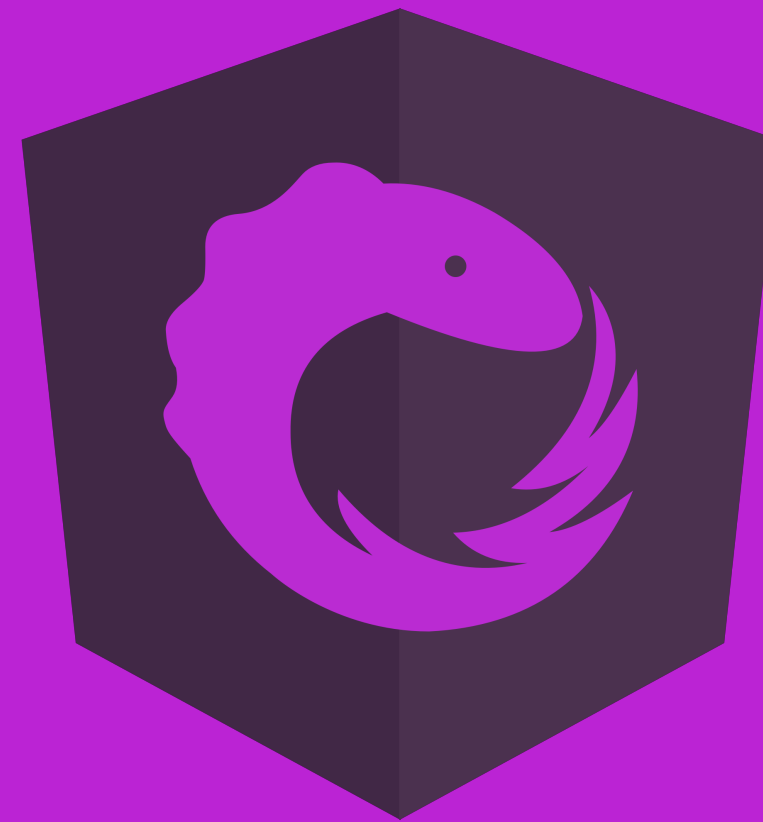
```
export function reducer(
  state: State = initialState,
  action: MovieApiActions.Union | MoviesPageActions.Union
): State {
  switch (action.type) {
    case MoviesPageActions.enter.type: {
      return { ...state, activeMovieId: null };
    }


    default: {
      return state;
    }
  }
}
```

```
export const moviesReducer = createReducer(
  initialState,
  on(MoviesPageActions.enter, state => ({
    ...state,
    activeMovieId: null
  }))
);
```

```javascript
export const moviesReducer = createReducer(
  initialState,
  on(MovieApiActions.createMovieSuccess, (state, action) =>
    adapter.addOne(action.movie, state)
  )
);
```
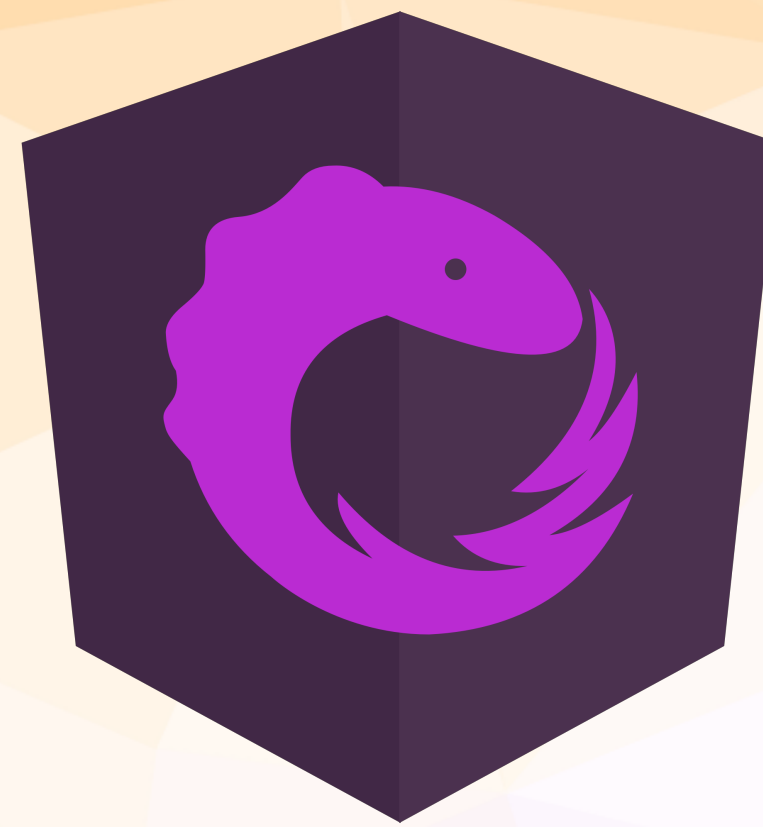
GOODBYE ACTION UNION TYPES!

```
export function reducer(state: State | undefined, action: Action) {
  return moviesReducer(state, action);
}
```

Demo

Challenge

1. In **books.reducer.ts** use **createReducer** and **on** to create a new **booksReducer** constant

2. Update the exported reducer in books.reducer.ts to use the new **booksReducer**

3. Delete the **action union types** from **books-page.actions.ts** and **books-api.actions.ts**

EFFECTS EXAMPLES

```
@Effect() tick$ = interval(/* Every minute */ 60 * 1000).pipe(
  map(() => Clock.tickAction(new Date()))
);
```

```
@Effect() = fromWebSocket("/ws").pipe(map(message => {
  switch (message.kind) {
    case "book_created": {
      return WebSocketActions.bookCreated(message.book);
    }


    case "book_updated": {
      return WebSocketActions.bookUpdated(message.book);
    }


    case "book_deleted": {
      return WebSocketActions.bookDeleted(message.book);
    }
  }
}))
```

```
@Effect()
createBook$ = this.actions$.pipe(
  ofType(BooksPageActions.createBook.type),
  mergeMap(action =>
    this.booksService.create(action.book).pipe(
      map(book => BooksApiActions.bookCreated({ book })),
      catchError(error => of(BooksApiActions.createFailure({
        error,
        book: action.book,
      })))
    )
  )
);
```

```
@Effect() promptToRetry$ = this.actions$.pipe(
  ofType(BooksApiActions.createFailure),
  mergeMap(action =>
    this.snackBar
      .open("Failed to save book.", "Try Again", {
        duration: /* 12 seconds */ 12 * 1000
      })
      .onAction()
      .pipe(
        map(() => BooksApiActions.retryCreate(action.book))
      )
  )
);
```

```typescript
@Effect() promptToRetry$ = this.actions$.pipe(
  ofType(BooksApiActions.createFailure),
  mergeMap(action =>
    this.snackBar
      .open("Failed to save book.", "Try Again", {
        duration: /* 12 seconds */ 12 * 1000
      })
      .onAction()
      .pipe(
        map(() => BooksApiActions.retryCreate(action.book))
      )
  )
);
```

| Failed to save book. | TRY AGAIN |

```
@Effect()
createBook$ = this.actions$.pipe(
  ofType(
    BooksPageActions.createBook,
    BooksApiActions.retryCreate,
  ),
  mergeMap(action =>
    this.booksService.create(action.book).pipe(
      map(book => BooksApiActions.bookCreated({ book })),
      catchError(error => of(BooksApiActions.createFailure({
        error,
        book: action.book,
      })))
    )
  )
);
```

```
@Effect({ dispatch: false })
openUploadModal$ = this.actions$.pipe(
  ofType(BooksPageActions.openUploadModal),
  tap(() => {
    this.dialog.open(BooksCoverUploadModalComponent);
  })
);
```

```typescript
@Effect() uploadCover$ = this.actions$.pipe(
  ofType(BooksPageActions.uploadCover),
  concatMap(action =>
    this.booksService.uploadCover(action.cover).pipe(
      map(result => BooksApiActions.uploadComplete(result)),
      takeUntil(
        this.actions$.pipe(
          ofType(BooksPageActions.cancelUpload)
        )
      )
    )
  )
);
```
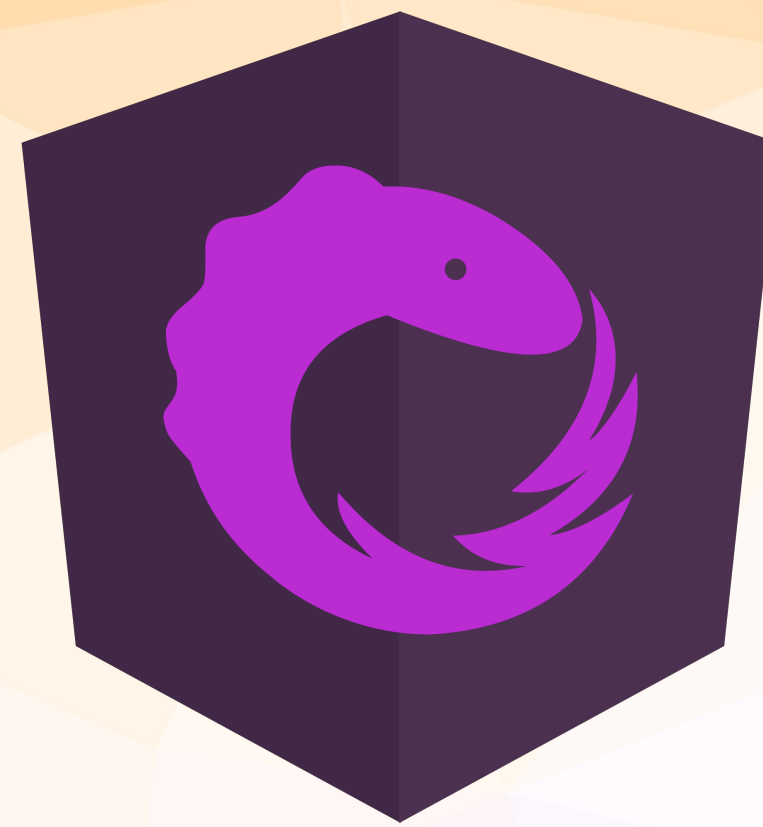
TESTING REDUCERS

```
it("should return the initial state when initialized", () => {
  const state = reducer(undefined, {
    type: "@@init"
  } as any);

  expect(state).toBe(initialState);
});
```

```typescript
const movies: Movie[] = [
  { id: "1", name: "Green Lantern", earnings: 0 }
];
const action = MovieApiActions.loadMoviesSuccess({
  movies
});

const state = reducer(initialState, action);
```

```typescript
const movie: Movie = {
  id: "1",
  name: "mother!",
  earnings: 1000
};
const firstAction = MovieApiActions.createMovieSuccess({ movie });
const secondAction = MoviesPageActions.deleteMovie({ movie });

const state = [firstAction, secondAction].reduce(
  reducer,
  initialState
);
```

```typescript
const movies: Movie[] = [
  { id: "1", name: "Green Lantern", earnings: 0 }
];
const action = MovieApiActions.loadMoviesSuccess({
  movies
});


const state = reducer(initialState, action);


expect(selectAllMovies(state)).toEqual(movies);
```

```
expect(state).toEqual({
  ids: ["1"],
  entities: {
    "1": { id: "1", name: "Green Lantern", earnings: 0 }
  }
});
```
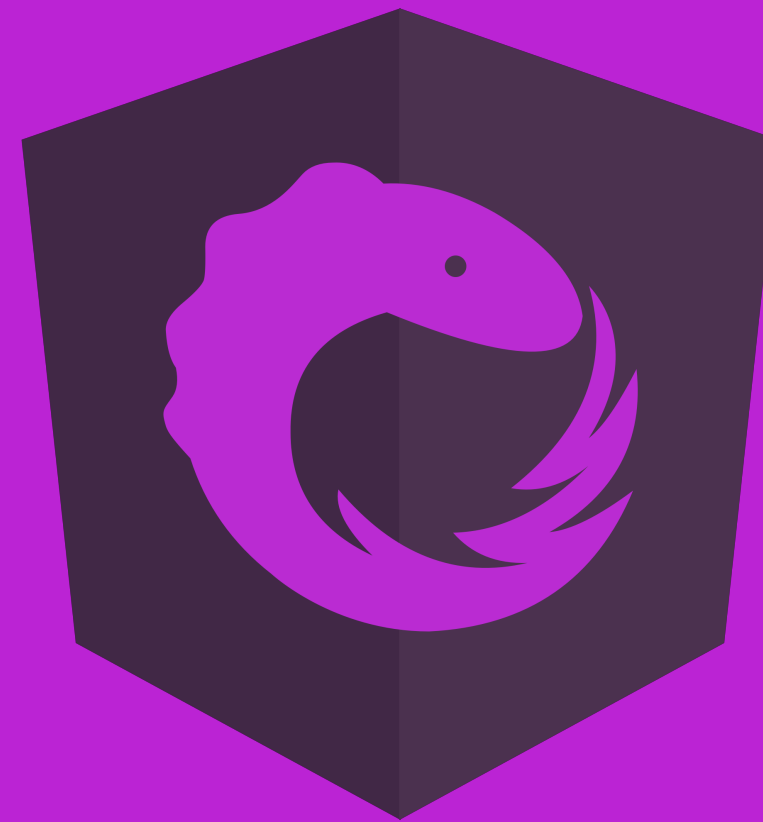
SNAPSHOT TESTING

```
expect(state).toMatchSnapshot();
```

in: **shared/state/__snapshots__/movie.reducer.spec.ts.snap**

```
exports[
  `Movie Reducer should load all movies when the API loads them all successfully 1`
] = `
Object {
  "activeMovieId": null,
  "entities": Object {
    "1": Object {
      "earnings": 0,
      "id": "1",
      "name": "Green Lantern",
    },
  },
  "ids": Array [
    "1",
  ],
}
`;
```
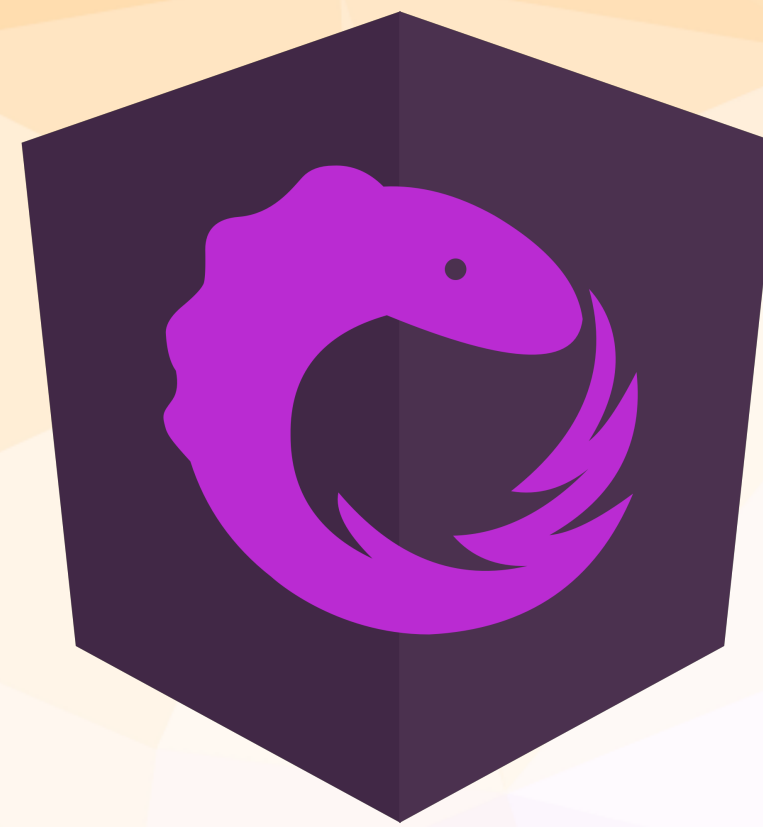
SNAPSHOT TESTING

- ✅ Avoid writing out manual assertions
- ✅ Verify how state transitions impact state
- ✅ Can be used with components
- ✅ Creates snap files that get checked in

Demo

Challenge

1. Write a test that verifies the **books reducer** returns the initial state when no state is provided using the **toBe** matcher
2. Write tests that verifies the **books reducer** correctly transitions state for **loading all books**, **creating a book**, and **deleting a book** using the **toMatchSnapshot** matcher
3. Write tests that verifies the behavior of the **selectActiveBook** and **selectAll** selectors
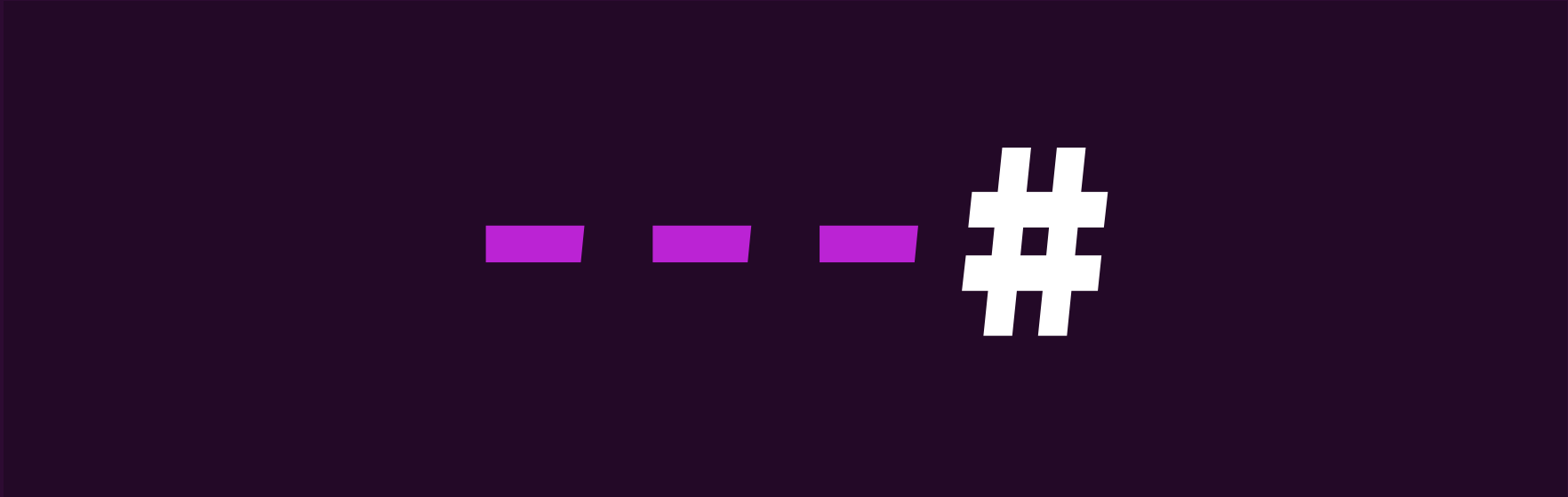
TESTING EFFECTS

OBSERVABLE TIMELINES

```
import { timer } from "rxjs";
import { mapTo } from "rxjs/operators";

timer(50).pipe(mapTo("a"));
```

```
timer(50).pipe(mapTo("a"));
```

-----a |

```
timer(30).pipe(mergeMap(() => throwError('Error!')))
```

---#

```
const source$ = timer(50).pipe(mapTo("a"));

const expected$ = cold("-----a|");


expect(source$).toBeObservable(expected$);
```

```
const source$ = timer(30).pipe(
  mergeMap(() => throwError("Error!"))
);
const expected$ = cold("---#", {}, "Error!");

expect(source$).toBeObservable(expected$);
```

–	10ms of time
a b c ...	Emission of any value
#	Error
|	Completion

COLD AND HOT OBSERVABLES

NETFLIX

Cable TV

`Actions`	Hot Observable
`HttpClient`	Cold Observables
`Store`	Hot Observable
`fromWebSocket`	Cold Observable

```typescript
let actions$: Observable<any>;

beforeEach(() => {
  TestBed.configureTestingModule({
    providers: [provideMockActions(() => actions$)]
  });
});


actions$ = hot("---a---", {
  a: BooksPageActions.enter
});
```

```javascript
const inputAction = MoviesPageActions.createMovie({
  movie: {
    name: mockMovie.name,
    earnings: 25
  }
});
const outputAction = MovieApiActions.createMovieSuccess({
  movie: mockMovie
});

actions$ = hot("--a---", { a: inputAction });
const response$ = cold("--b|", { b: mockMovie });
const expected$ = cold("----c--", { c: outputAction });
mockMovieService.create.mockReturnValue(response$);

expect(effects.createMovie$).toBeObservable(expected$);
```
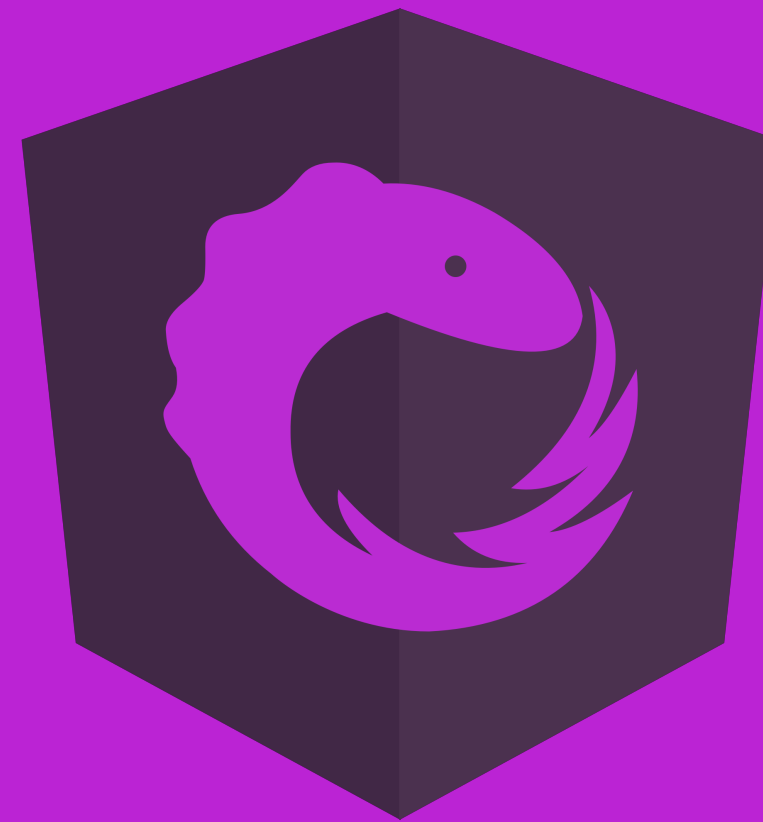
JASMINE MARBLES

- ☑ Make assertions about time

- ☑ Describe Rx behavior with diagrams

- ☑ Verify observables behave as described

- ☑ Works with hot and cold observables

Challenge

1. Open **books-api.effects.spec.ts** and declare variables for the **actions$**, instance of the **effects**, and a mock **bookService**
2. Use the **TestBed** to setup **providers** for the effects, actions, and the book service
3. Verify the behavior of the **createBook$** effect using mock actions and test observables
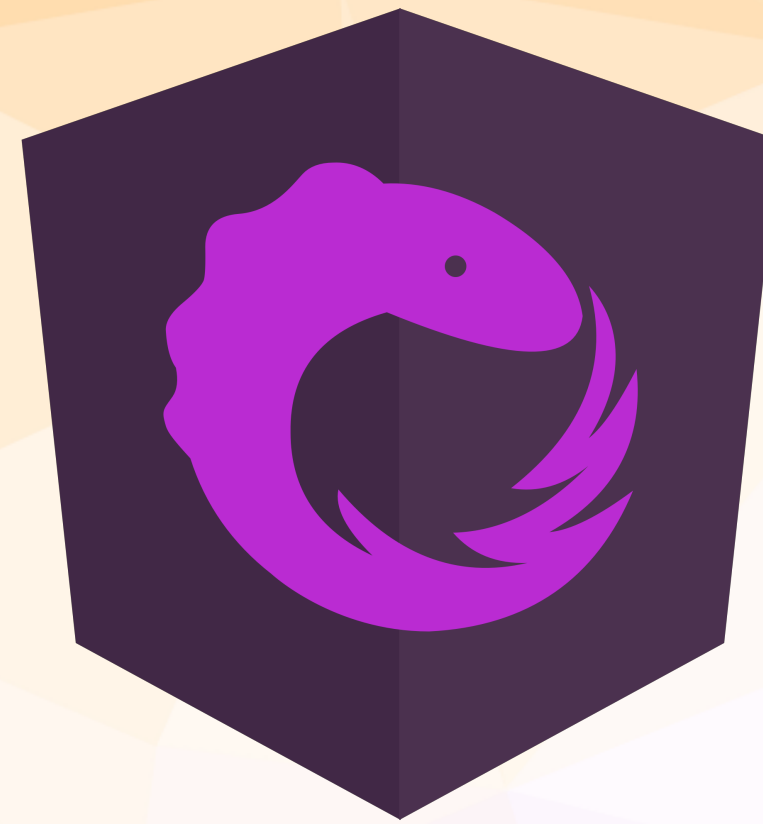
FOLDER LAYOUT

LIFT

- ✅ Locating our code is easy

- ✅ Identify code at a glance

- ✅ Flat file structure for as long as possible

- ✅ Try to stay DRY - don't repeat yourself

```
src/
  shared/
    // Shared code

  modules/
    ${feature}/
      // Feature code
```

```
modules/
  ${feature}/
    actions/
      ${action-category}.actions.ts
      index.ts

    components/
      ${component-name}/
        ${component-name}.component.ts
        ${component-name}.component.spec.ts

    services/
      ${service-name}.service.ts
      ${service-name}.service.spec.ts

    effects/
      ${effect-name}.effects.ts
      ${effect-name}.effects.spec.ts

    ${feature}.module.ts
```

```
modules/
  book-collection/
    actions/
      books-page.actions.ts
      index.ts

    components/
      books-page/
        books-page.component.ts
        books-page.component.spec.ts

    services/
      books.service.ts
      books.service.spec.ts

    effects/
      books.effects.ts
      books.effects.spec.ts

    book-collection.module.ts
```

ACTION BARRELS

```
import * as BooksPageActions from "./books-page.actions";
import * as BooksApiActions from "./books-api.actions";

export { BooksPageActions, BooksApiActions };
```

```
import { BooksPageActions } from "app/modules/book-collection/actions";
```

```
src/
  shared/
    state/
      ${state-name}/
        ${state-key}/
          ${state-key}.reducer.ts
          ${state-key}.spec.ts
          index.ts

        ${feature-name}.state.ts
        ${feature-name}.state.spec.ts
        ${feature-name}.state.module.ts
        index.ts

    effects/
      ${effect-name}/
        ${effect-name}.effects.ts
        ${effect-name}.effects.spec.ts
        ${effect-name}.actions.ts
        ${effect-name}.module.ts
        index.ts
```

```
src/
    shared/
        state/
            core/
                books/
                    books.reducer.ts
                    books.spec.ts

                core.state.ts
                core.state.spec.ts
                core.state.module.ts
                index.ts

        effects/
            clock/
            clock.effects.ts
            clock.effects.spec.ts
            clock.actions.ts
            clock.module.ts
```

FOLDER STRUCTURE

- ☑ Put state in a shared place separate from features

- ☑ Effects, components, and actions belong to features

- ☑ Some effects can be shared

- ☑ Reducers reach into modules' action barrels

"How does NgRx work?"

"How does NgRx work?"

State flows down, changes flow up

☑ State flows down, changes flow up

☑ Indirection between state & consumer

- ✅ State flows down, changes flow up
- ✅ Indirection between state & consumer
- ✅ Select & Dispatch => Input & Output

- ☑ State flows down, changes flow up
- ☑ Indirection between state & consumer
- ☑ Select & Dispatch => Input & Output
- ☑ Adhere to single responsibility principle

STORE

☑ State contained in a single state tree

☑ State in the store is immutable

☑ Slices of state are updated with reducers

REDUCERS

- ✅ Produce new states

- ✅ Receive the last state and next action

- ✅ Switch on the action type

- ✅ Use pure, immutable operations

ACTIONS

- ✅ Unified interface to describe events

- ✅ Just data, no functionality

- ✅ Has at a minimum a type property

- ✅ Strongly typed using createAction

ENTITY

- ☑ Working with collections should be fast

- ☑ Collections are very common

- ☑ Common set of basic state operations

- ☑ Common set of basic state derivations

EFFECTS

- ✅ Processes that run in the background

- ✅ Connect your app to the outside world

- ✅ Often used to talk to services

- ✅ Written entirely using RxJS streams

SNAPSHOT TESTING

- ☑ Avoid writing out manual assertions
- ☑ Verify how state transitions impact state
- ☑ Can be used with components
- ☑ Creates snap files that get checked in

JASMINE MARBLES

- ☑ Make assertions about time

- ☑ Describe Rx behavior with diagrams

- ☑ Verify observables behave as described

- ☑ Works with hot and cold observables

"How does NgRx work?"

Yaaaaaaaaaaaaaaaaay!

HELP US IMPROVE

https://bit.ly/2ROXvn0

FOLLOW ON TALKS

"Good Action Hygiene" by Mike Ryan

https://youtu.be/JmnsEvoy-gY

"Reactive Testing Strategies with NgRx" by Brandon Roberts & Mike Ryan

https://youtu.be/MTZprd9tI6c

"Authentication with NgRx" by Brandon Roberts

https://youtu.be/46IRQgNtCGw

"You Might Not Need NgRx" by Mike Ryan

https://youtu.be/omnwu_etHTY

"Just Another Marble Monday" by Sam Brennan & Keith Stewart

https://youtu.be/dwDtMs4mN48

@ngrx/schematics

@ngrx/schematics

@ngrx/router-store

@ngrx/schematics

@ngrx/router-store

@ngrx/data

@ngrx/schematics

@ngrx/router-store

@ngrx/data

ngrx.io

QUESTIONS?

@MikeRyanDev

@brandontroberts

THANK YOU