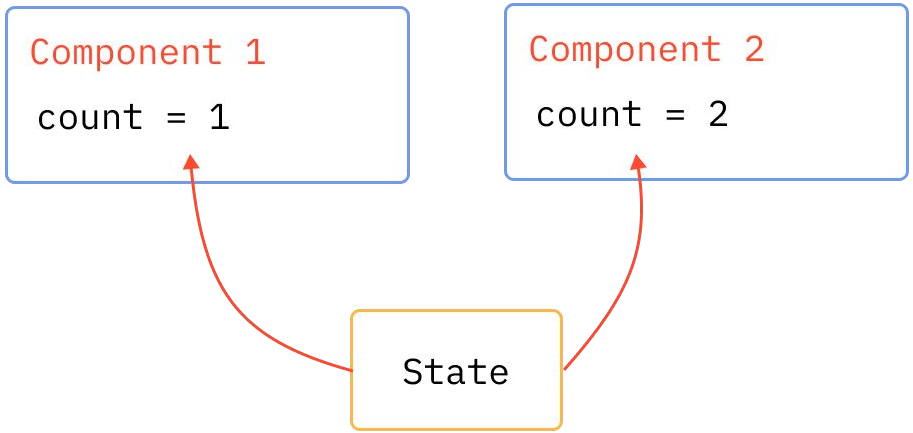
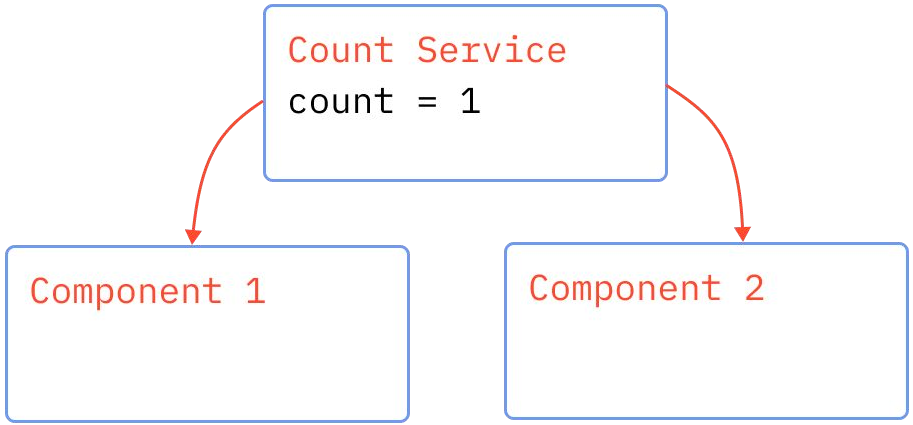
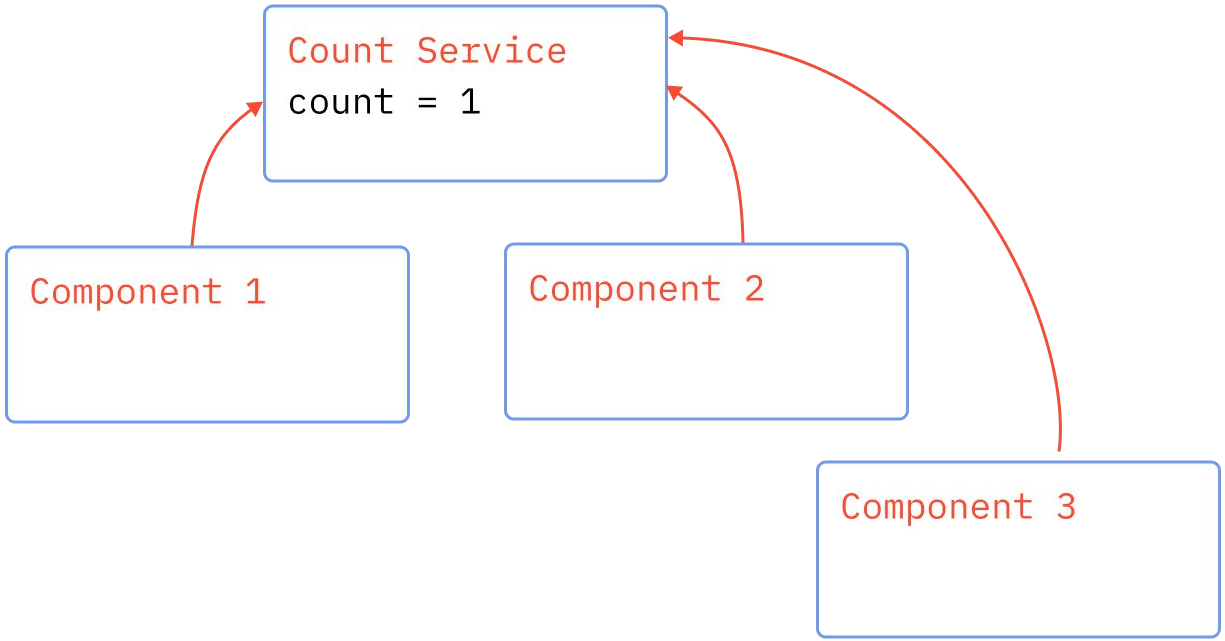
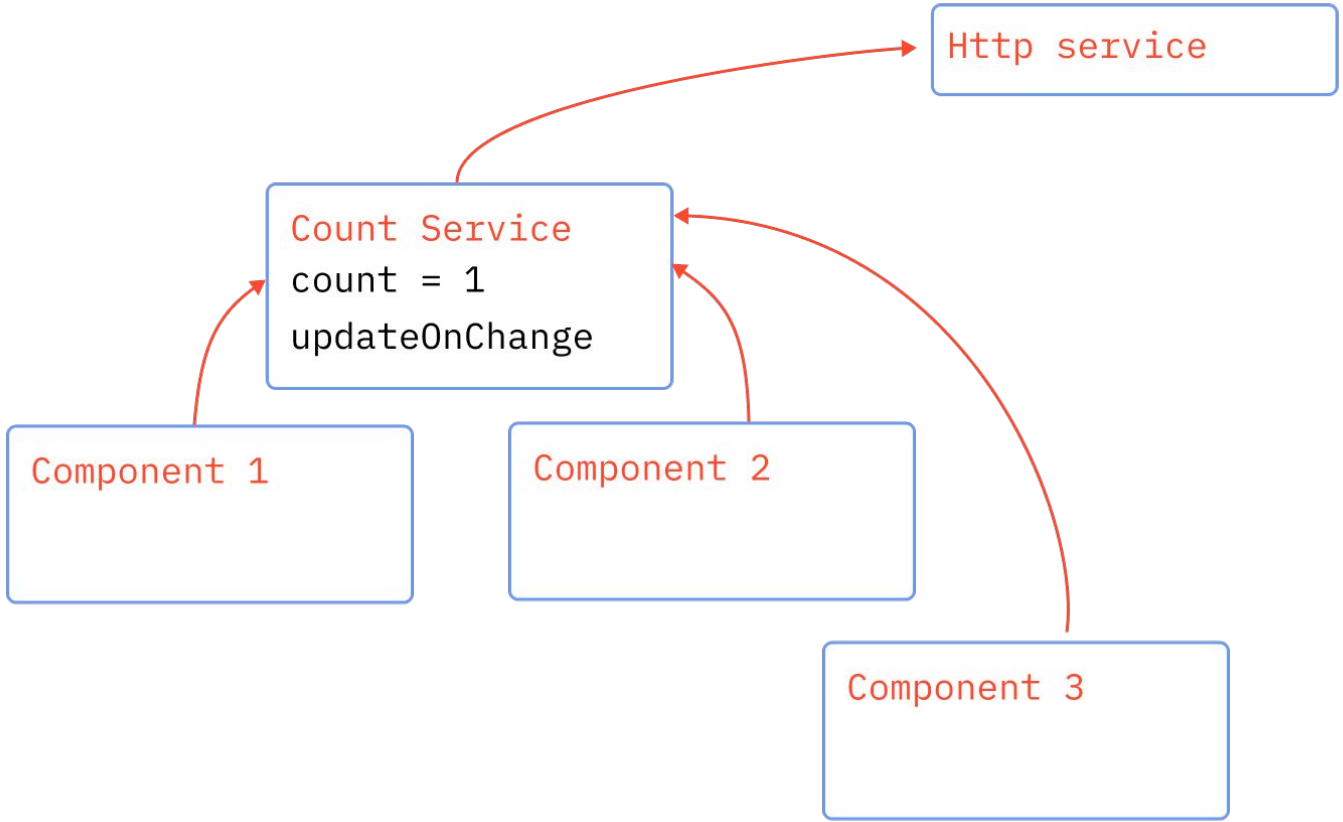
State Management

Лектор: Петър Маламов









State Management

Процес на управление на state-а на приложението, който включва съхраняване, актуализиране и синхронизиране на данни между различни компоненти.

Осигурява последователност и предсказуемост на поведението на приложението, улеснявайки взаимодействието между компонентите и поддържайки цялостната структура на приложението.

NgRx

Какво е NgRx?

Библиотека за управление на **state** в Angular приложения, базиран на принципите на **Redux** и **RxJS**.

Предоставя централизирано хранилище за данни, управление на асинхронни операции и обработка на странични ефекти (side effects).

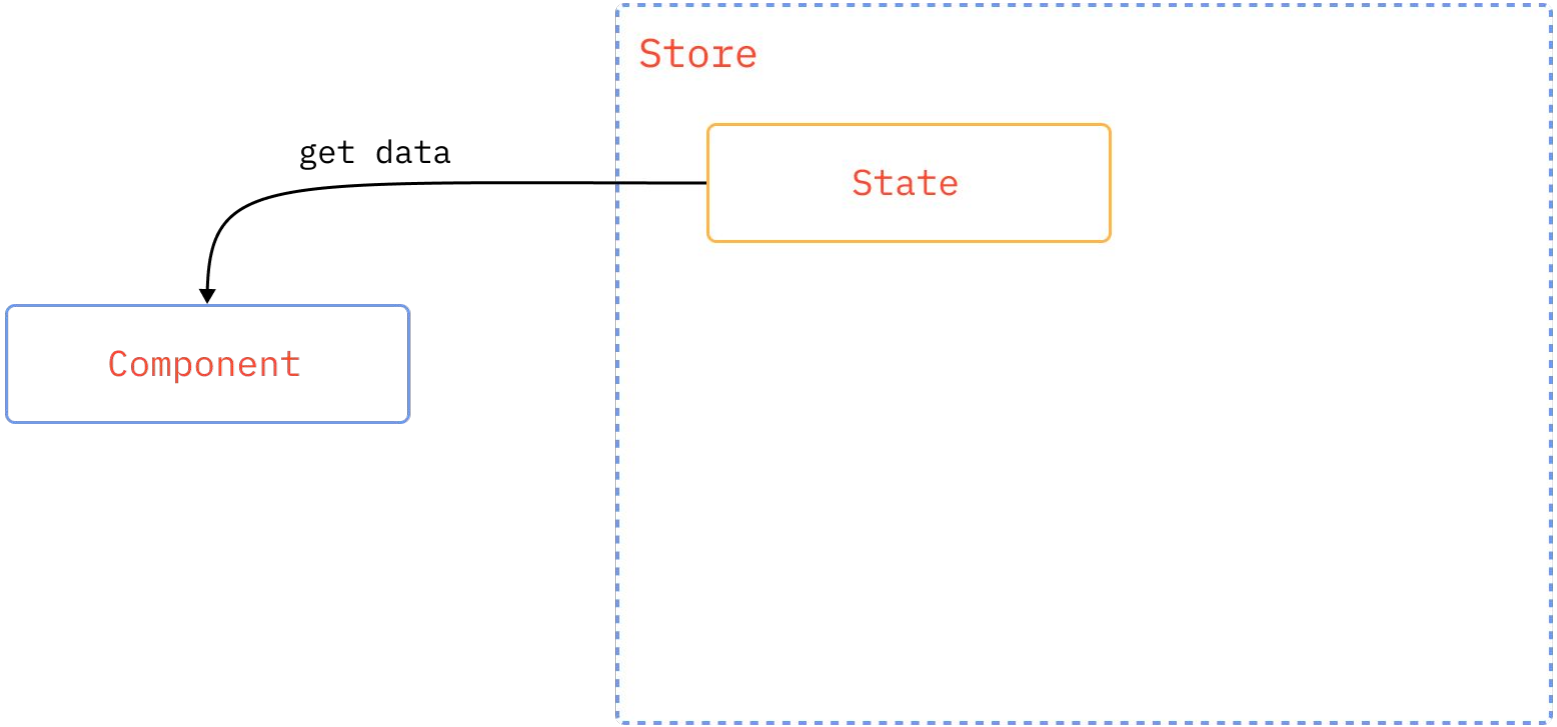
Основни концепции на NgRx

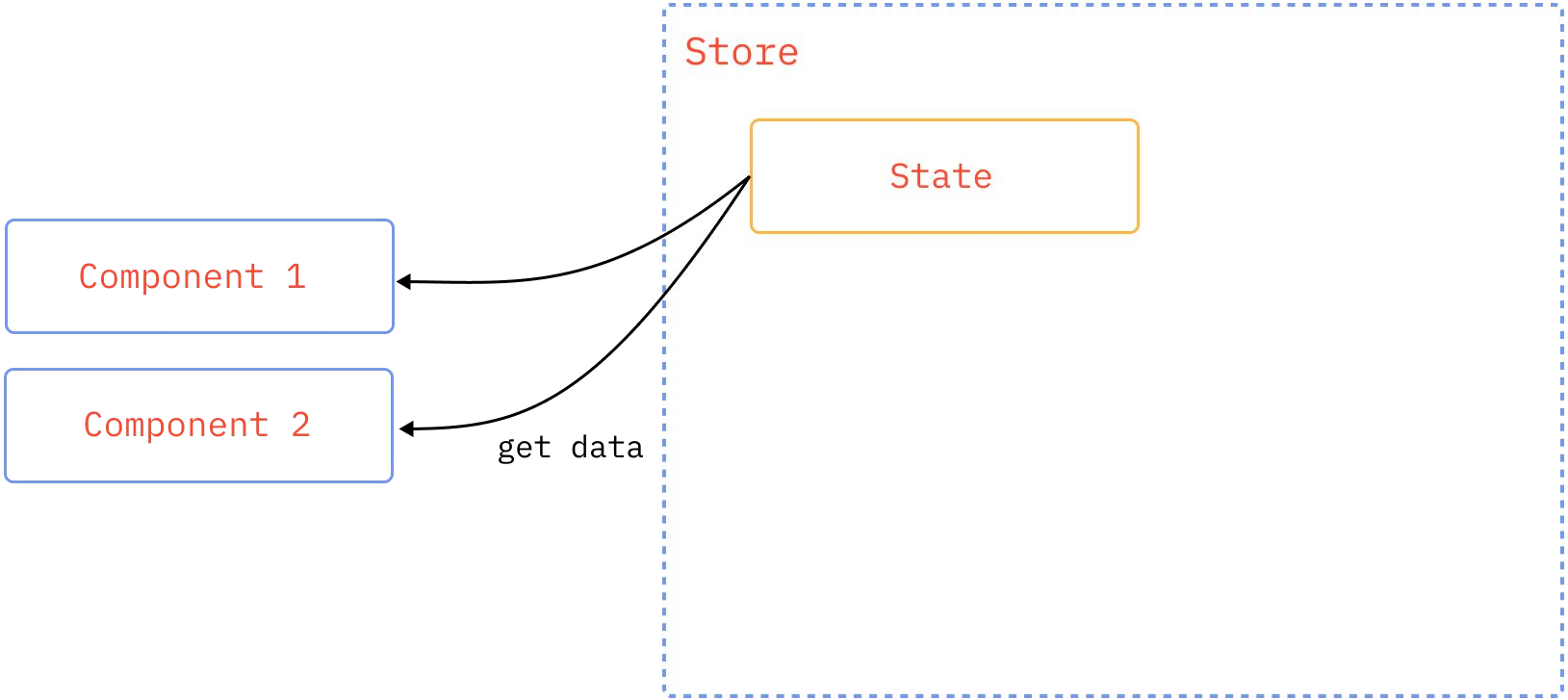
- Store
- Actions
- Reducers
- Selectors

Store

Централизирано хранилище, което съхранява **state**-а на приложението на едно място.

Позволява на различни компоненти да споделят и достъпват общи данни, като поддържа "**единствения източник на истина**" в приложението.





Важно!

Данните в хранилището трябва да бъдат immutable, което означава, че не бива да се променят директно.

Actions

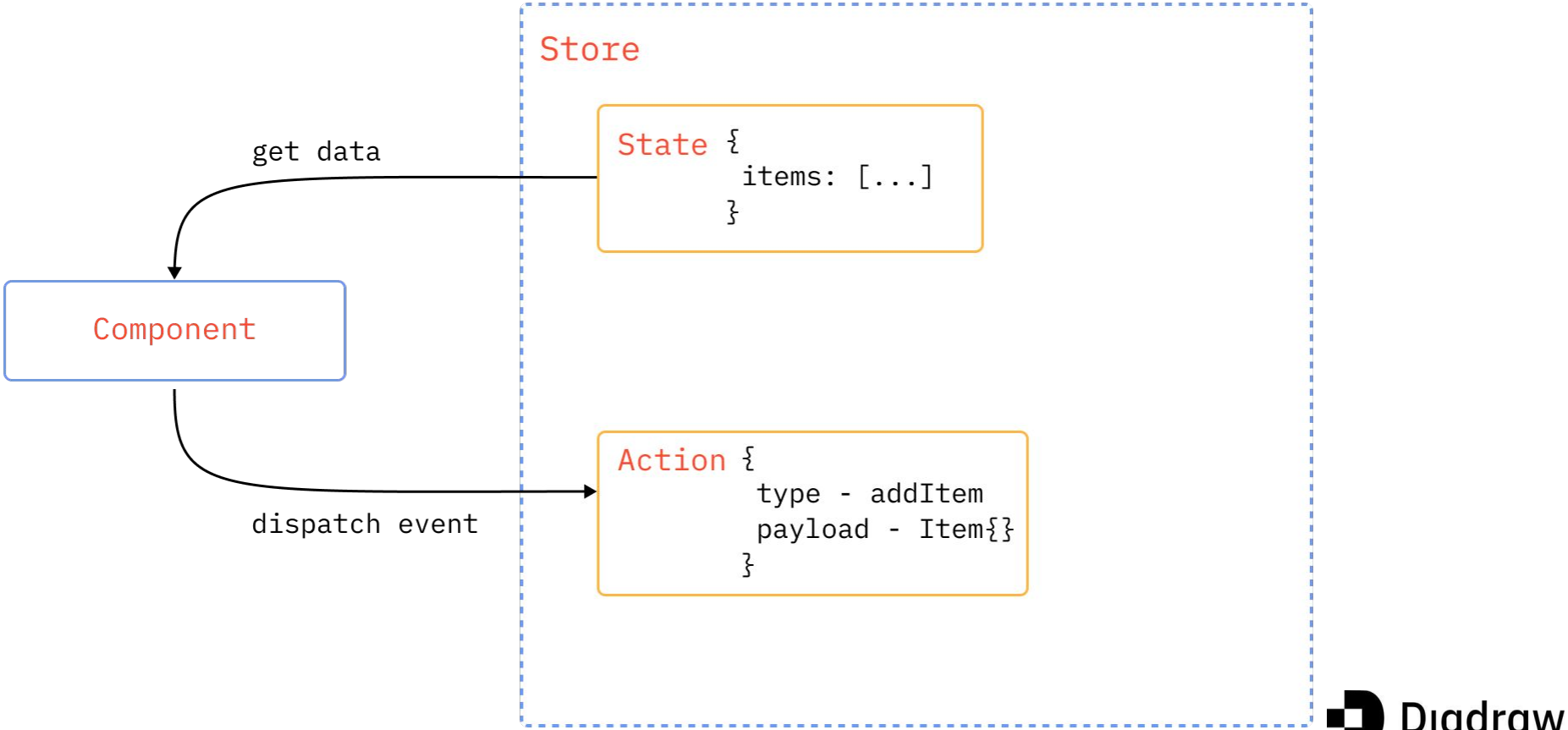
Обекти, които описват намерение за промяна на state в приложението.

За да уведомим хранилището за желаната промяна, трябва да изпратим Action-а, използвайки метода **dispatch** на Store-а.

Actions съдържат:

- Тип на промяната - низ, който описва вида на промяната, която искаме да извършим в Store-а.

- Данни, необходими за извършване на промяната (**опционални**)



Reducers

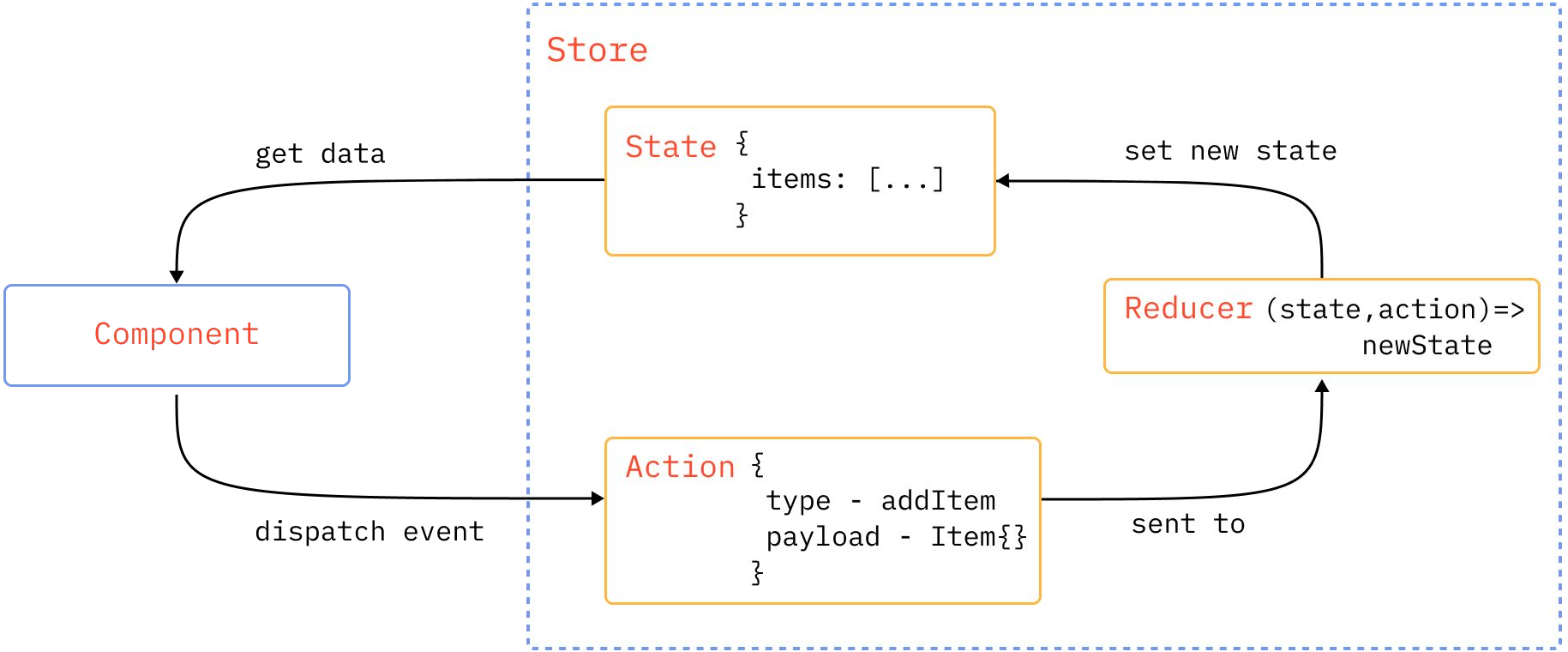
Функции, които определят как се променя състоянието на приложението в отговор на изпратени Actions.

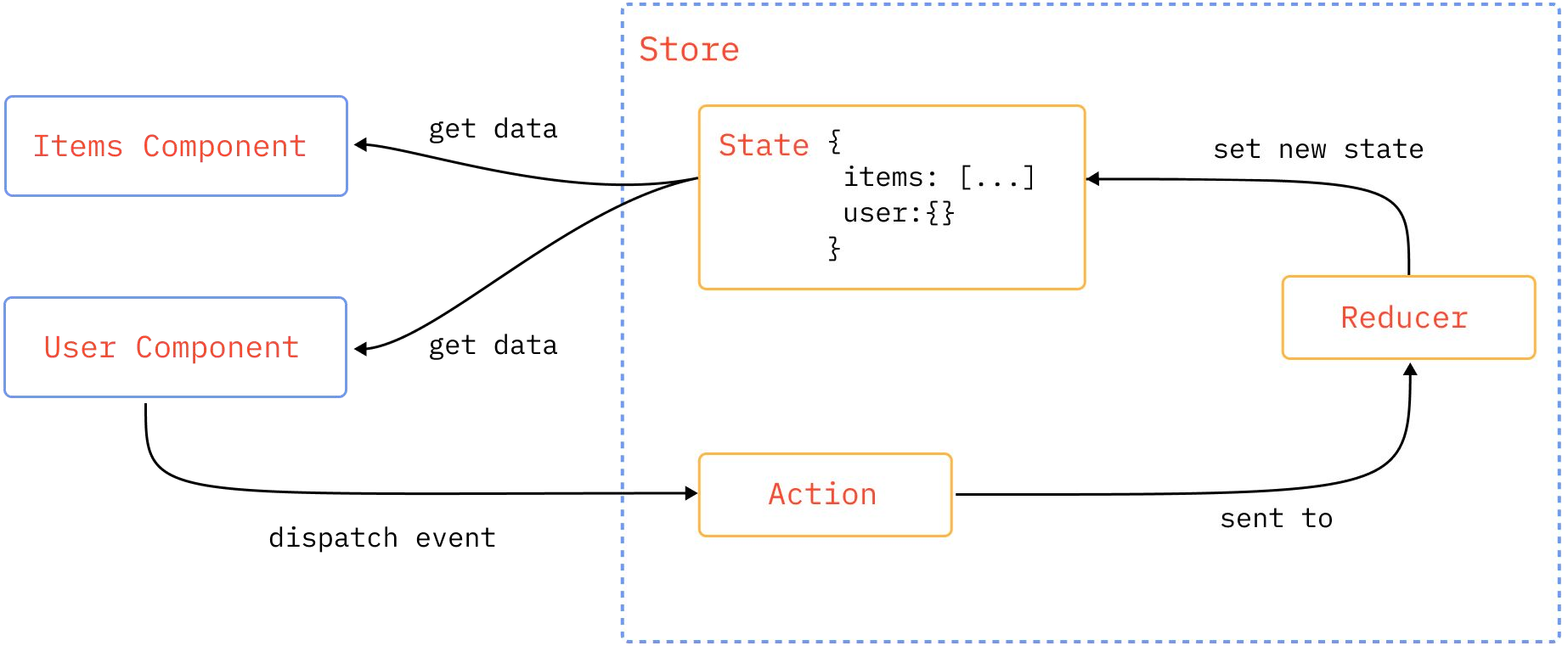
Приемат текущото състояние и Action като аргументи и връщат ново състояние.

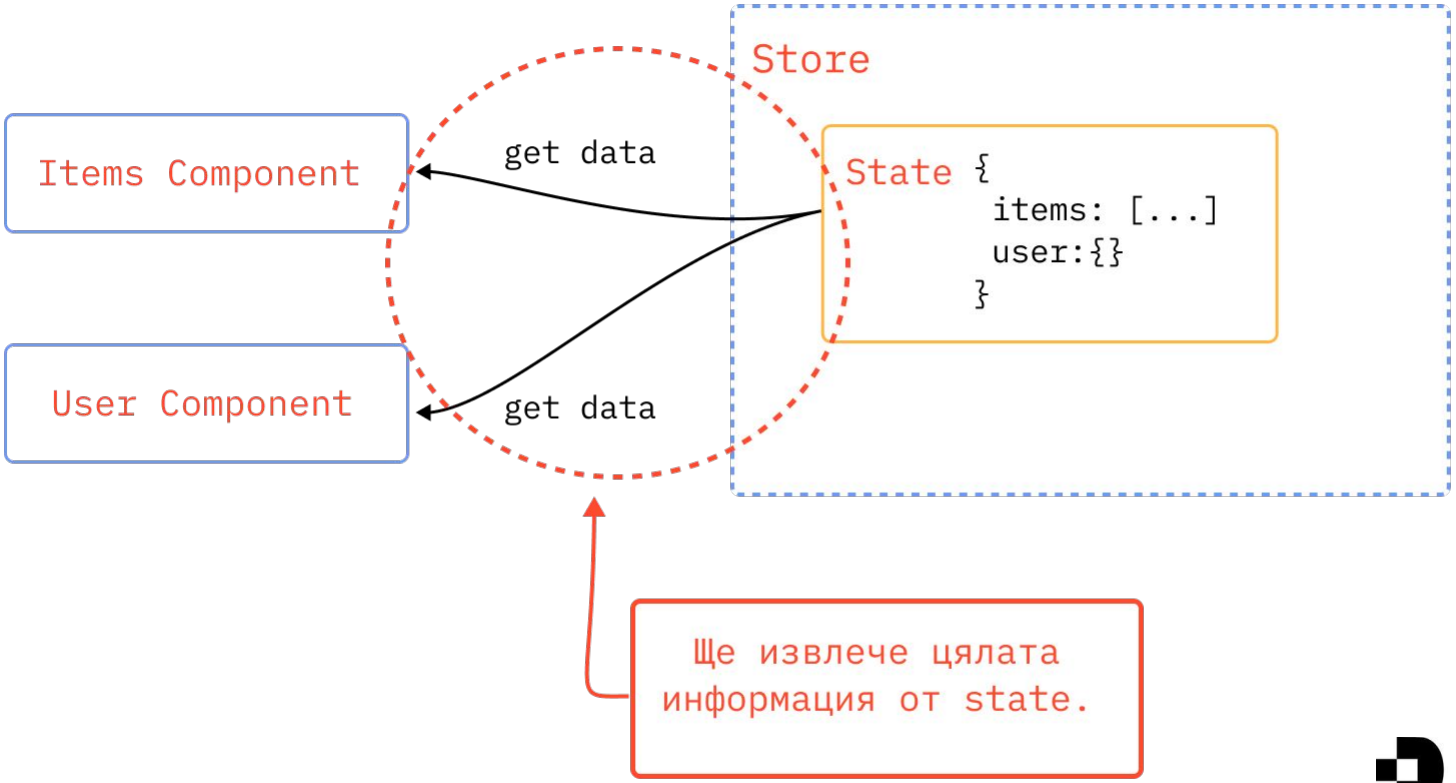
Вместо да модифицират съществуващото състояние, те създават ново!

Важно!

Reducers функциите трябва да са **pure** - при еднакви входни аргументи, винаги трябва да връща един и същи резултат.





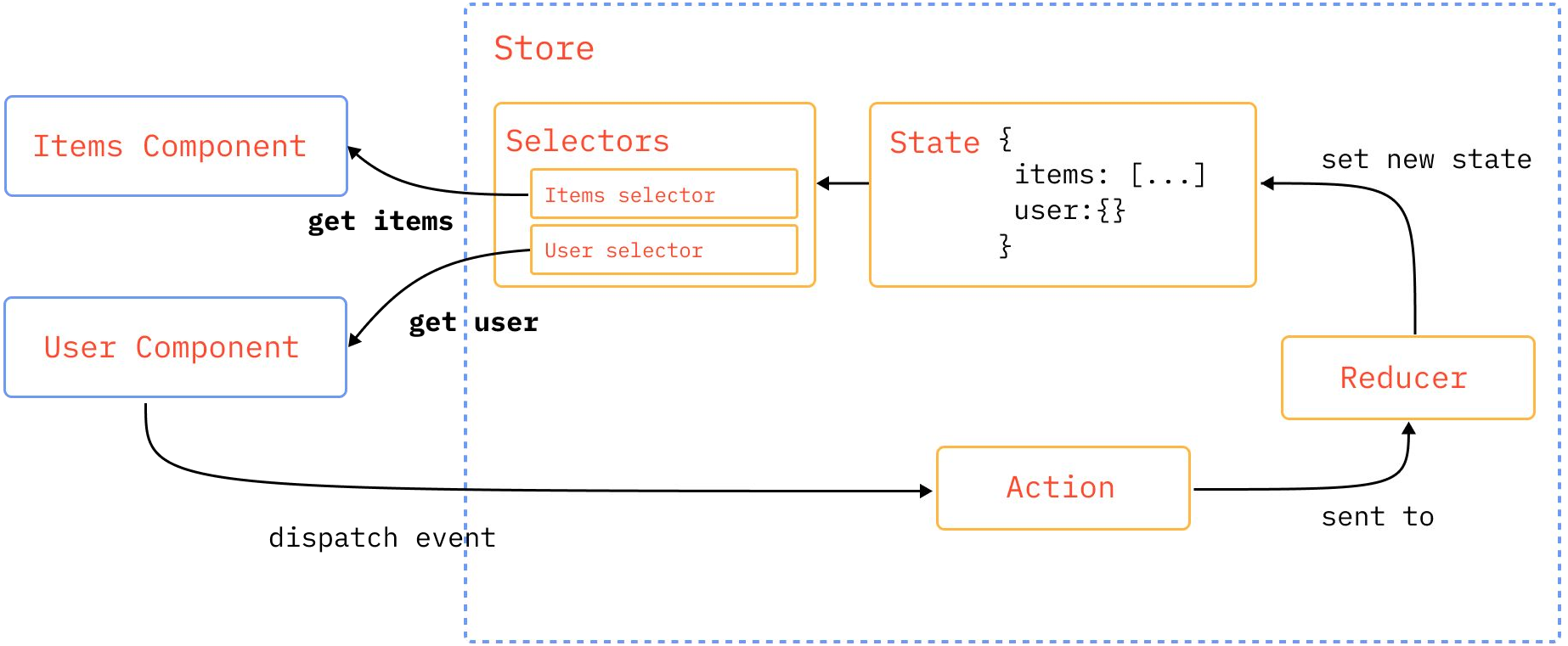


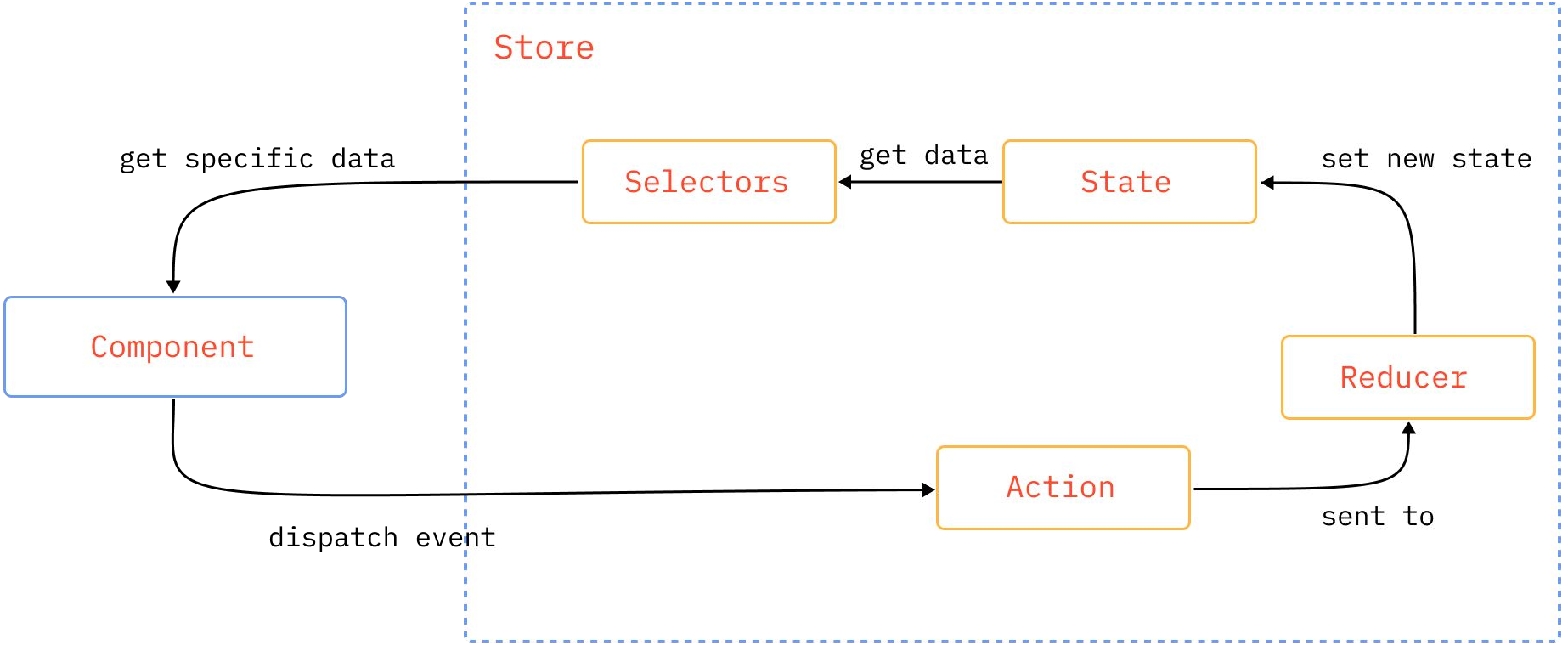
Selectors

Функции, които се използват за извличане на **специфични части** от store-а.

Selectors се използват за достъп до конкретни части от Store-а, което позволява компонентите да получават само необходимата информация, без да се налага да работят с цялото състояние.

Подобно на **reducers**, **selectors** трябва да са **pure** функции!





Redux pattern

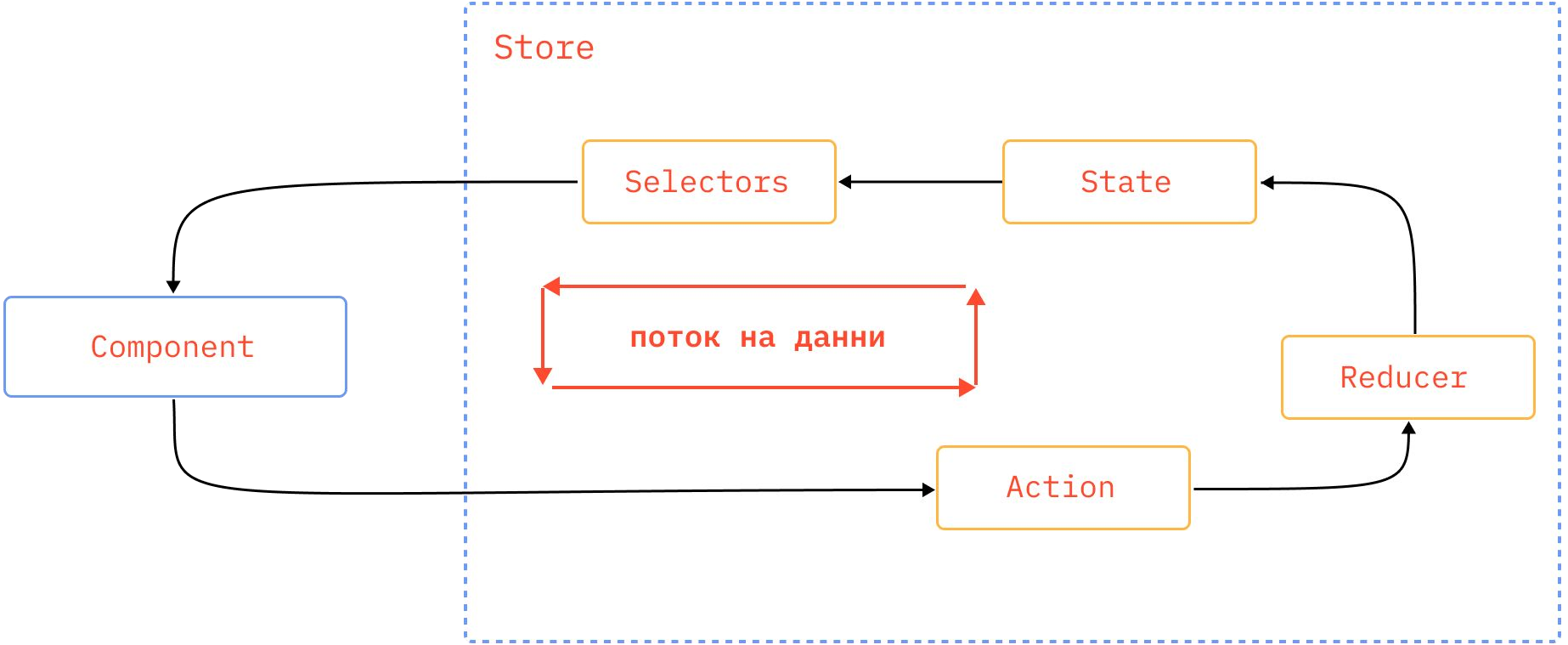
Какво е Redux pattern?

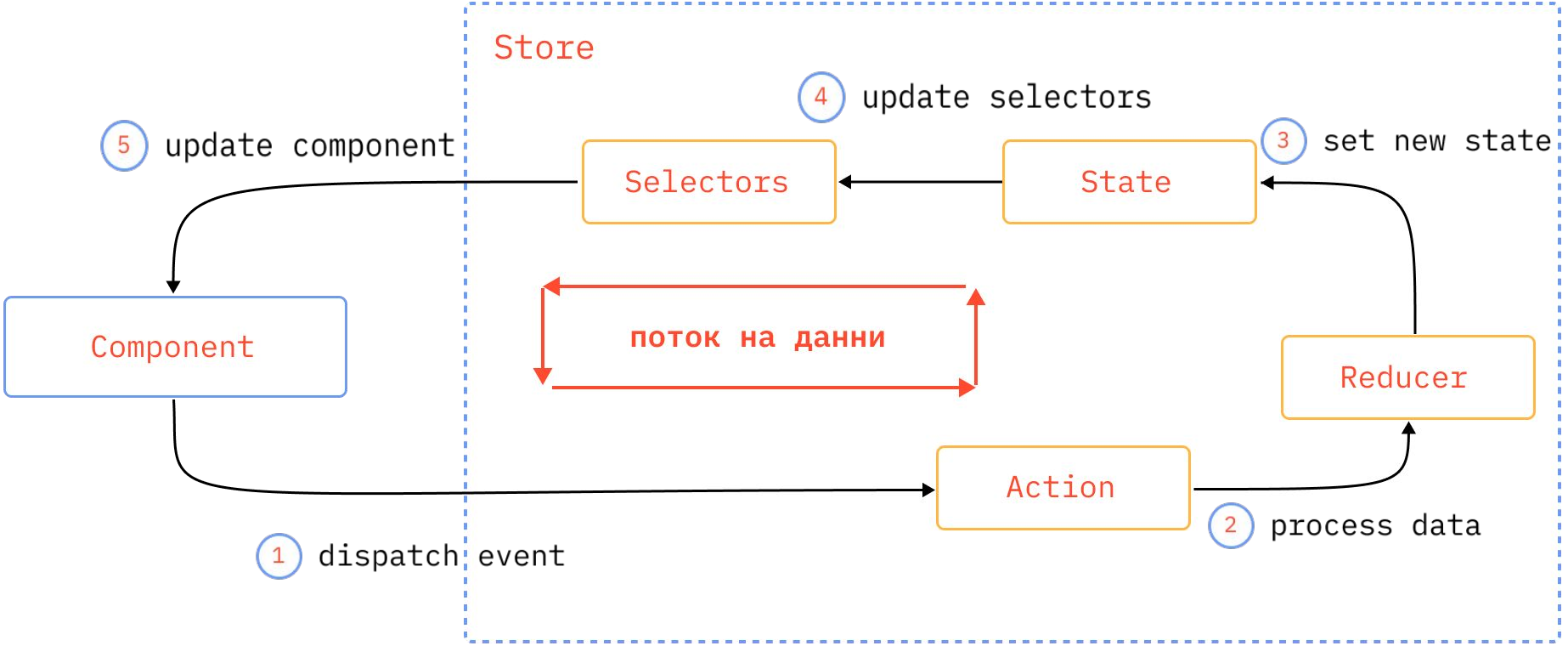
Архитектура, която улеснява управлението на state в големи приложения.

Основната идея на **Redux** е, че целият state на приложението се съхранява в едно централизирано хранилище (**Store**), което улеснява достъпа до данни, координацията и предсказуемостта на приложението.

Важно!

Redux следва **еднопосочен поток на данни**, което означава, че данните преминават през приложението в една посока, което прави промяната на състоянието и поведението му лесно предсказуемо и управляемо.





Ползи от Redux pattern

- Предсказуемост на действията.
- Централизирано хранилище за данните.
- Скалируемост.
- По-лесно дебъгване и проследяване на грешки.

Негативи от Redux pattern

- Увеличаване на boilerplate-та.
- Повишена сложност.
- Learning Curve.

Кога да използваме Redux Pattern?

- Много компоненти споделят едни и същи данни
- Приложението има сложна логика за промяна на данни
- Работа с много асинхронни операции
- Голям екип работи по приложението

NgRx в Angular

Създаване на Store

TS app.config.ts

```ts
export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes), provideStore()],
};
```

Създаване на Action

За да създадем **Action** в NgRx, трябва да използваме функцията **createAction**.

При създаването на **Action** е задължително да му зададем тип, а по избор можем да му подадем и допълнителни данни (payload).

TS counter.actions.ts

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

export const increment = createAction('[Counter Component] Increment');

export const add = createAction(
  '[Counter Component] Add',
  props<{ value: number }>()
);
```

Използване Action

TS counter.actions.ts

```ts
export class CounterComponent {
  constructor(private store: Store<{ counter: number }>) {}

  increment() {
    this.store.dispatch(increment());
  }
}
```

Правила при създаване на Actions

- Разделяйте Actions по категории (според източника на събитията)
- Дефинирате Actions преди да изградите функционалността.
- Създавайте достатъчно Actions.
- Event-Driven
- Правете Actions описателни

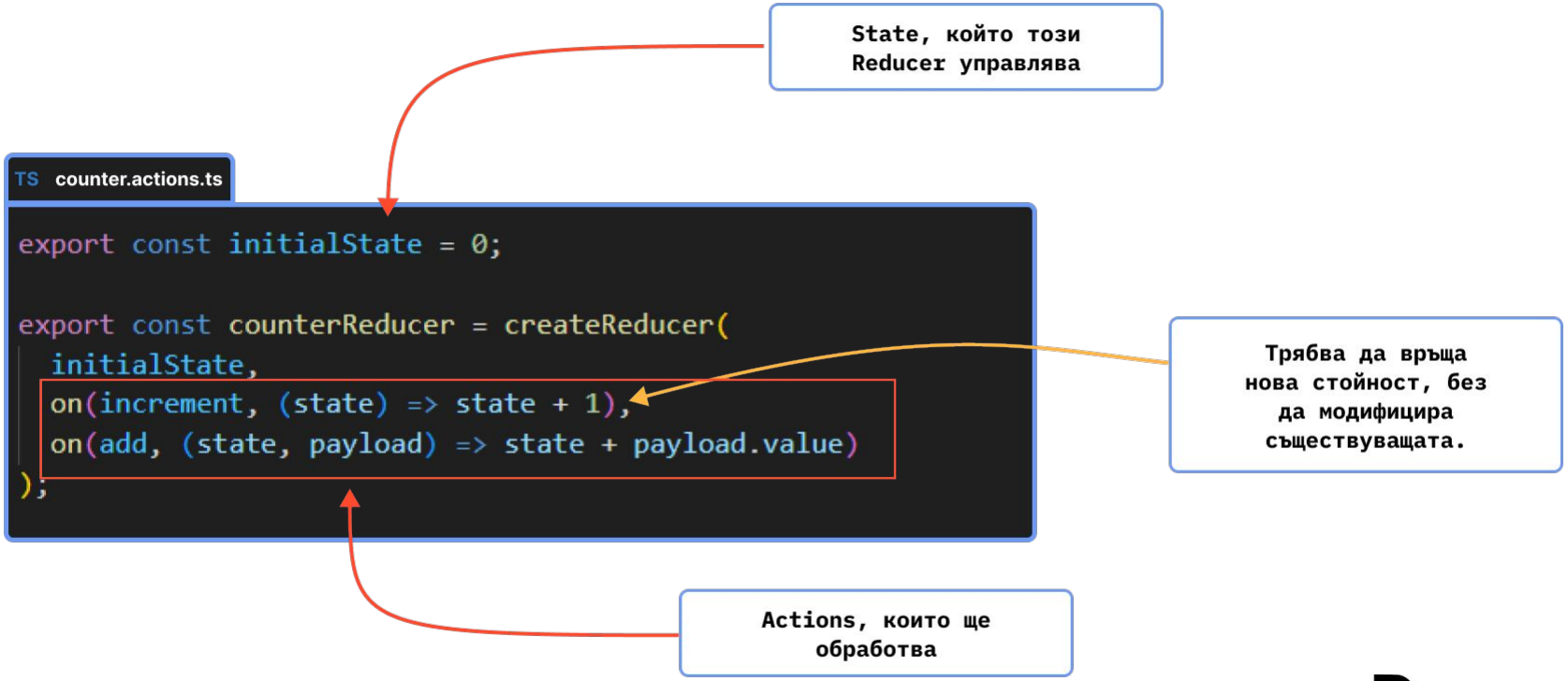
Създаване на Reducer

Всеки reducer обработва определени действия (**actions**) и управлява **конкретна част от state-а на приложението**, като връща ново състояние въз основа на полученото действие.

Той игнорира непознати действия и се грижи единствено за своята част от цялостното състояние.

За да създадем **Reducer** в NgRx, трябва да използваме функцията **createReducer**.

Използване на Reducer



Важно!

Когато се изпрати action, всички регистрирани reducers го получават. Дали ще обработят този action, се определя от on функциите, които свързват едно или повече actions с определена промяна в state-а.

Регистриране на reducer.

За да може един reducer да обработва даден action и да променя съответния state, първо трябва да го регистрираме в store-а.

Има два основни типа регистрация:

- **Root регистрация** - гарантира, че всички reducers са дефинирани още при стартирането на приложението.

- **Feature регистрация** - reducers-те ще бъдат добавени към глобалния state само когато съответната функционалност бъде използвана.

TS app.config.ts

```ts
export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes),
    provideStore(),
    provideState({ name: 'counter', reducer: counterReducer }),
  ],
};
```

Root регистрация

TS app.routes.ts

```ts
export const routes: Routes = [
  {
    path: 'counter',
    loadComponent: () =>
      import('./counter/counter.component').then((m) => m.CounterComponent),
    providers: [provideState({ name: 'counter', reducer: counterReducer })],
  },
];
```

Не е задължително
да е lazy-loaded

Feature регистрация

Създаване на Selector

Всеки **Selector** е pure функция, която отговаря за извличането на конкретна част (slice) от state-а.

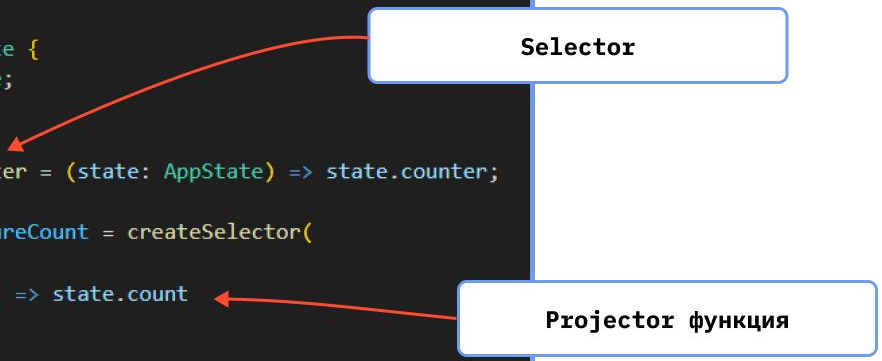
За да създадем **Selector** в NgRx, трябва да използваме функцията **createSelector**.

createSelector приема като аргументи:

- Входни селектори
- Projector функция - обработва резултатите от входните селектори и връща крайния резултат на селектора.

TS counter.selectors.ts

```ts
export interface SelectorState {
  count: number;
}

export interface AppState {
  counter: SelectorState;
}

export const selectCounter = (state: AppState) => state.counter;

export const selectFeatureCount = createSelector(
  selectCounter,
  (state: SelectorState) => state.count
);
```



Използване на Selector

TS counter.component.ts

```ts
export class CounterComponent {
  count$: Observable<number> = this.store.select(selectCount);

  constructor(private store: Store<AppState>) {}
}
```

Важно!

Функцията createSelector може да приема няколко selector-а като аргументи. Това улеснява достъпа до по-вложени данни или данни, които се намират в различни части на state-а.

По този начин може да се комбинира информация от няколко източника, за да се получи по-комплексен резултат.

TS selectors.ts

```ts
const appState: AppState = {
  orders: [],
  selectedCustomers: customers,
};

export const selectCustomer = (state: AppState) => state.selectedCustomers;
export const selectOrders = (state: AppState) => state.orders;


export const selectCustomerOrders = createSelector(
  selectCustomer,
  selectOrders,
  (selectedCustomer: Customer, orders: Order[]) => {
    // do some logic here
  }
);
```

Feature Selection

Освен функцията **createSelector**, NgRx предлага и **createFeatureSelector** за създаването на селектори.

Тази функция улеснява селектирането на конкретен feature state от store-а, което опростява достъпа до необходимата информация.

TS selectors.ts

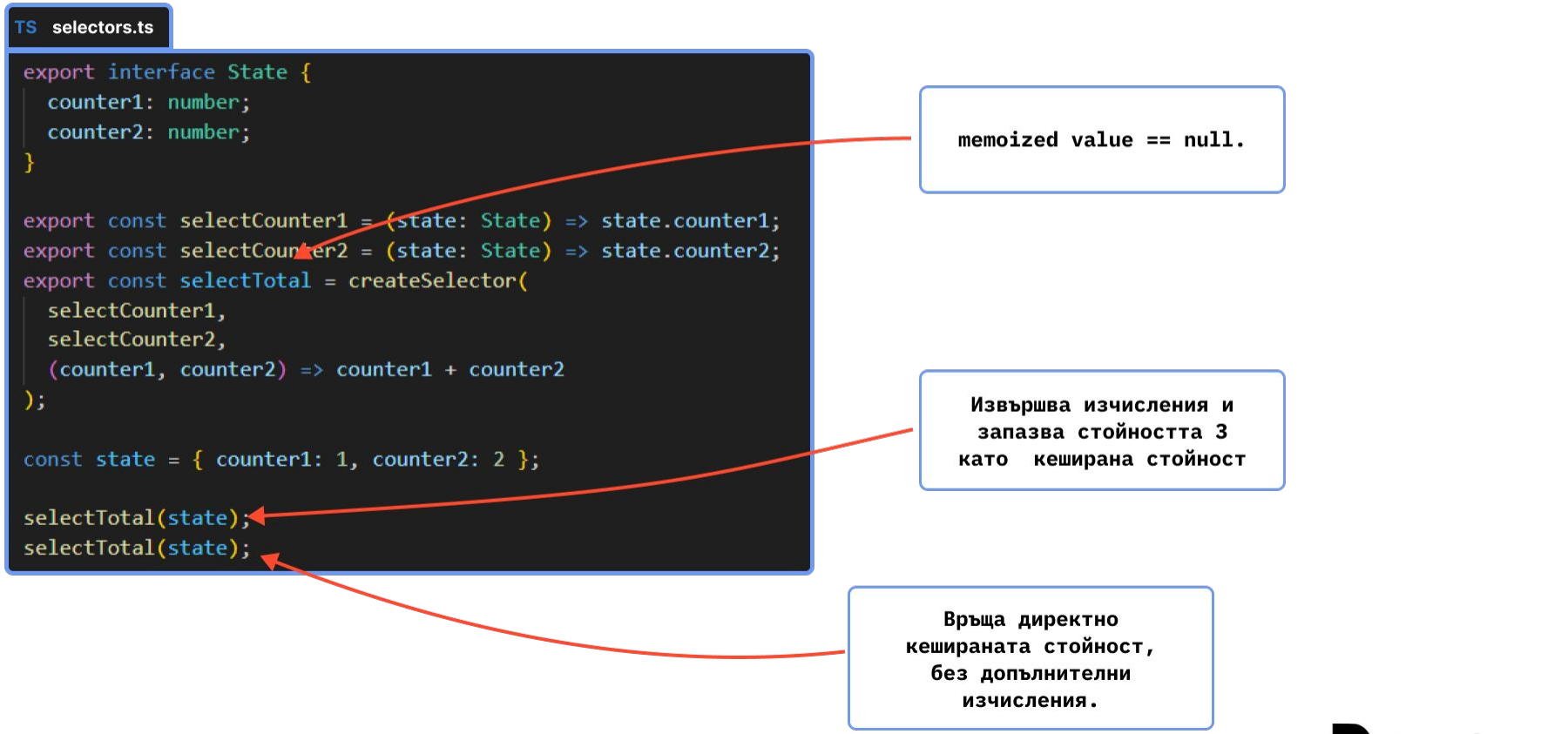
```ts
export const counterFeatureKey = 'counter';

export interface CounterState {
  count: number;
}

export const selectCounter =
  createFeatureSelector<CounterState>(counterFeatureKey);

export const selectCount = createSelector(
  selectCounter,
  (state: CounterState) => state.count
);
```

Селектиране на конкретен feature

Важно!

При използването на функциите createSelector и createFeatureSelector, NgRx следи стойностите, които селекторите връщат. Ако при следващо извикване аргументите са същите, функцията няма да се изпълни отново, а ще върне кеширана стойност - **memoization**.

При първоначалното създаване, createSelector и createFeatureSelector имат **memoized value == null**. След първото извикване върнатата стойност се запазва като кеширана стойност (**memoized value)**.



Премахване на кешираната стойност

Кешираната стойност на селектора остава в паметта за неопределено време, което може да доведе до съхраняване на голям обем от данни, които вече не са необходими.

За да освободим паметта можем да използваме метода **release** на selector-а.

TS selectors.ts

```ts
export interface State {
  counter1: number;
  counter2: number;
}

export const selectCounter1 = (state: State) => state.counter1;
export const selectCounter2 = (state: State) => state.counter2;
export const selectTotal = createSelector(
  selectCounter1,
  selectCounter2,
  (counter1, counter2) => counter1 + counter2
);

const state = { counter1: 1, counter2: 2 };

selectTotal(state);

selectTotal.release();
```

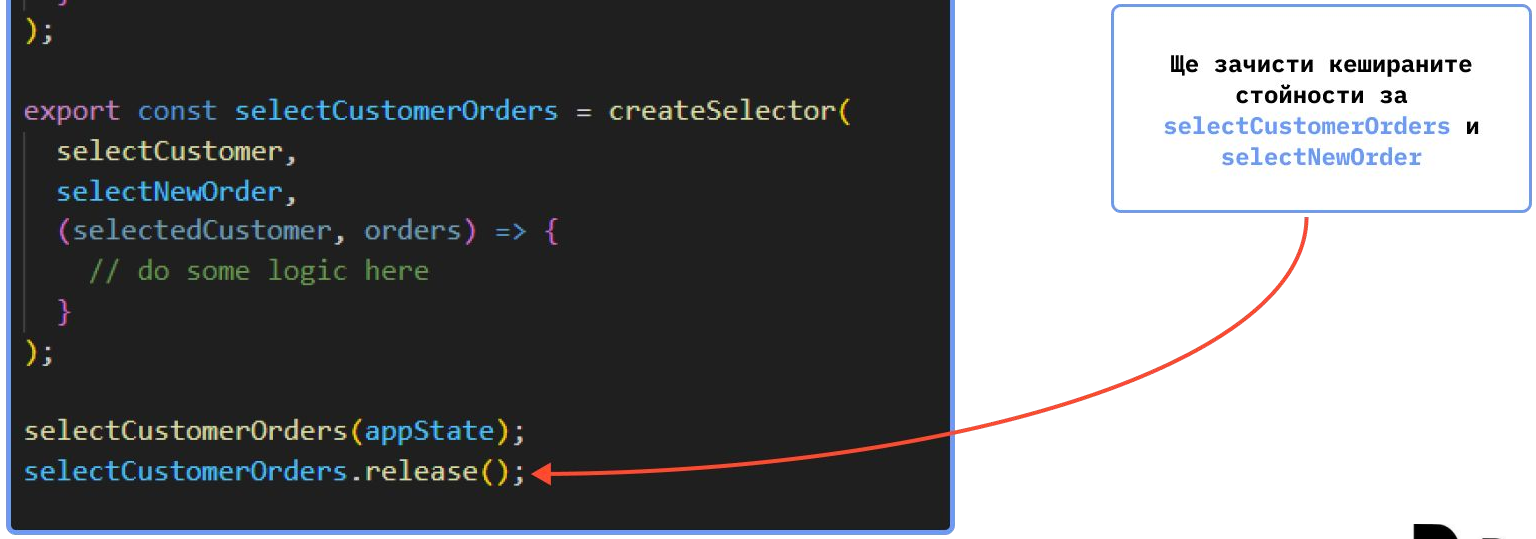
Важно!

Ако при създаването на селектора сме използвали няколко select функции, при премахване на кешираната стойност на главния селектор ще бъдат изчистени и всички свързани селектори, от които той зависи.

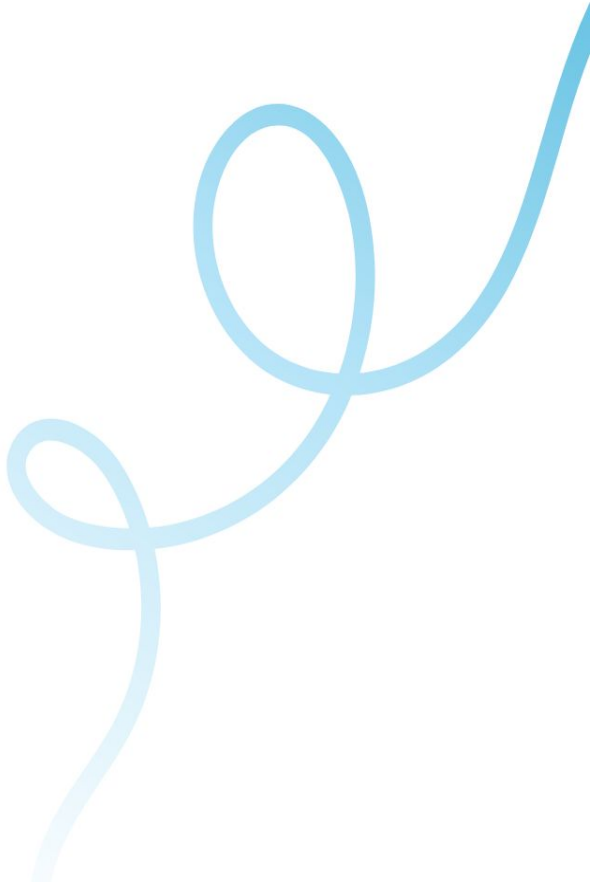
TS selectors.ts

```ts
export const selectNewOrder = createSelector(
  selectOrders,
  (orders: Order[]) => {
    // do some logic here
    return orders;
  }
);

export const selectCustomerOrders = createSelector(
  selectCustomer,
  selectNewOrder,
  (selectedCustomer, orders) => {
    // do some logic here
  }
);

selectCustomerOrders(appState);
selectCustomerOrders.release();
```

Ще зачисти кешираните стойности за `selectCustomerOrders` и `selectNewOrder`



Упражнение



Feature Creators

Функции, които опростяват създаването на selectors и reducers за определени части от state-а на приложението.

Те се използват за:

- организиране на кода
- намаляване на повторяемостта
- по-добра структура при функционалности (features) на приложението.

Създаване на Feature

За да създадем feature, използваме функцията **createFeature**, на която подаваме името на feature-а и съответния reducer.

След като създадем feature, функцията ще върне обект, който включва името на feature-а, reducer и selectors за всяка част от неговия state.

Създаване на Feature

TS orders.state.ts

```ts
const initialState = {
  orders: [],
  loading: false,
};

const ordersFeature = createFeature({
  name: 'orders',
  reducer: createReducer<OrdersState>(
    initialState,
    on(loadOrders, (state) => ({ ...state, loading: true }))
  ),
});

export const {
  name, // feature name
  reducer, // feature reducer
  selectOrdersState, // feature selector
  selectLoading, // selector for `loading` property
  selectOrders, // selector for `orders` property
} = ordersFeature;
```

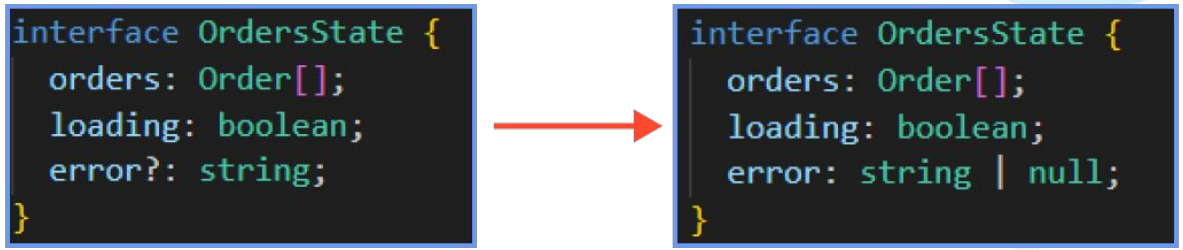
Използване на Feature

TS app.config.ts

```ts
export const routes: Routes = [
  {
    path: 'orders',
    providers: [provideState(ordersFeature)],
  },
];
```

Важно!

Когато използваме createFeature за създаване на feature, не може да имаме опционални части в state-а; всички свойства на state-а трябва да имат зададени начални стойности.



Създаване на допълнително Selectors

TS orders.state.ts

```ts
export const ordersFeature = createFeature({
  name: 'orders',
  reducer: createReducer<OrdersState>(
    initialState,
    on(loadOrders, (state) => ({ ...state, loading: true }))
  ),
  extraSelectors: ({ selectOrders, selectQuery }) => ({
    selectFilteredOrders: createSelector(
      selectOrders,
      selectQuery,
      (orders, query) => orders.filter((order) => order.id === query)
    ),
  }),
});
```

TS orders.state.ts

```ts
export const {
  name,
  reducer,
  selectOrdersState,
  selectLoading,
  selectOrders,

  selectFilteredOrders,
} = ordersFeature;
```

Благодаря за вниманието!