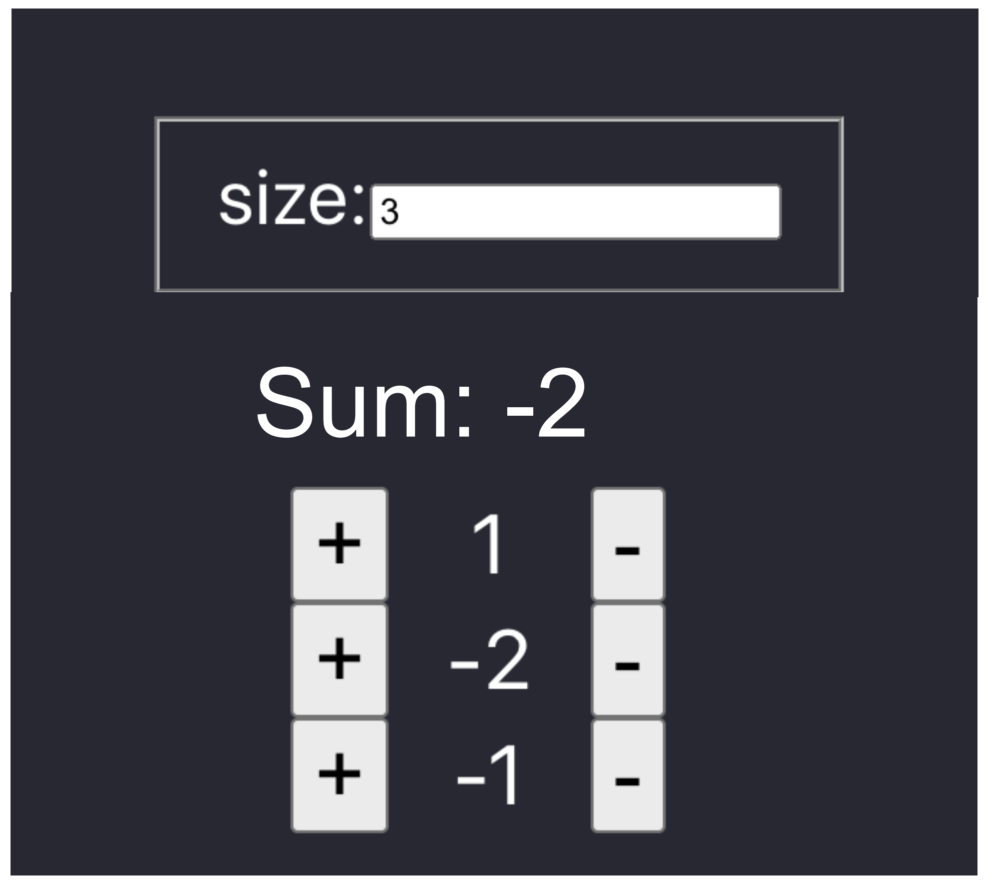
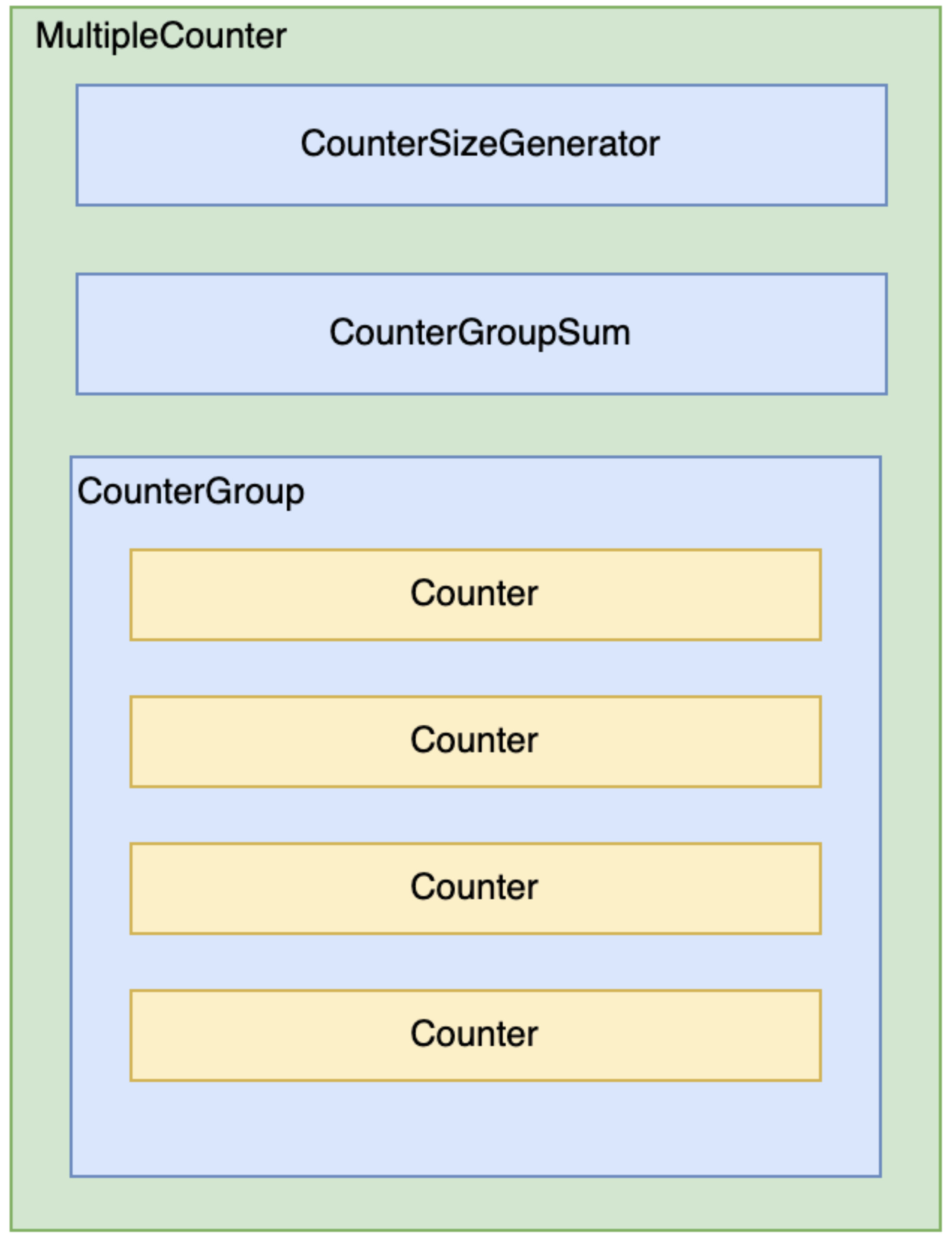
AFS
Agile Full Stack
Developer Bootcamp
Thoughtworks ©

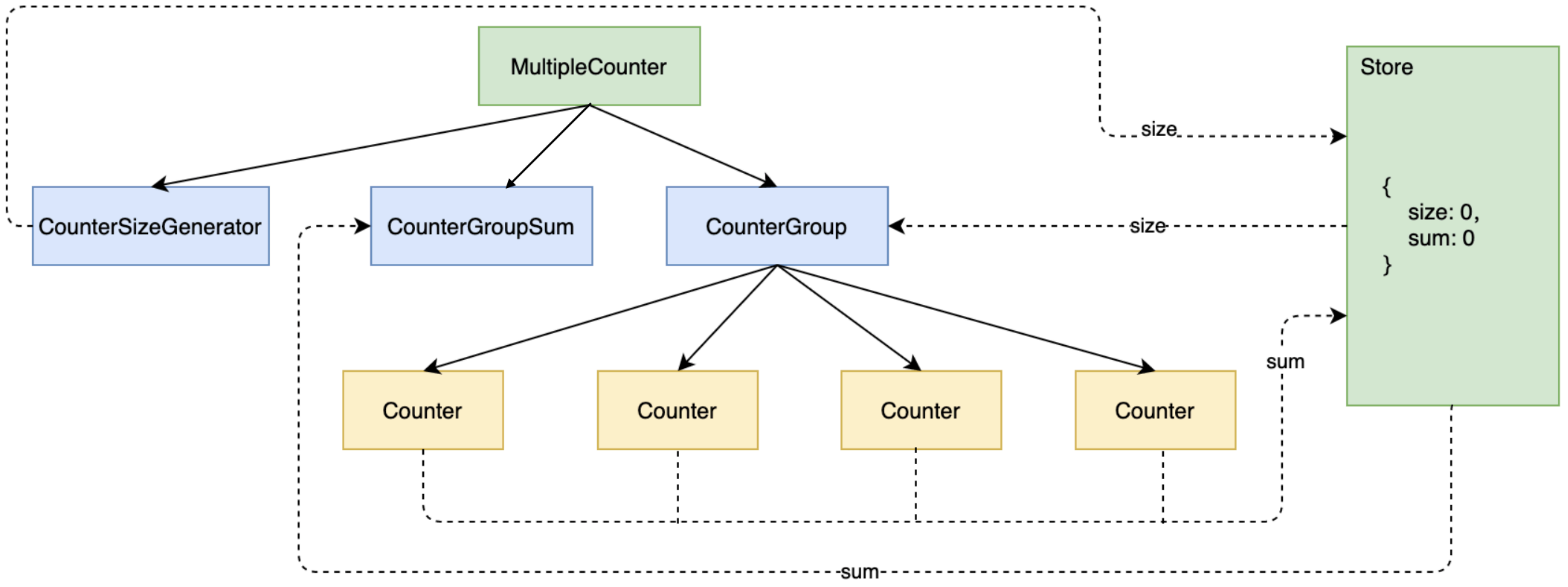
React redux

thoughtworks

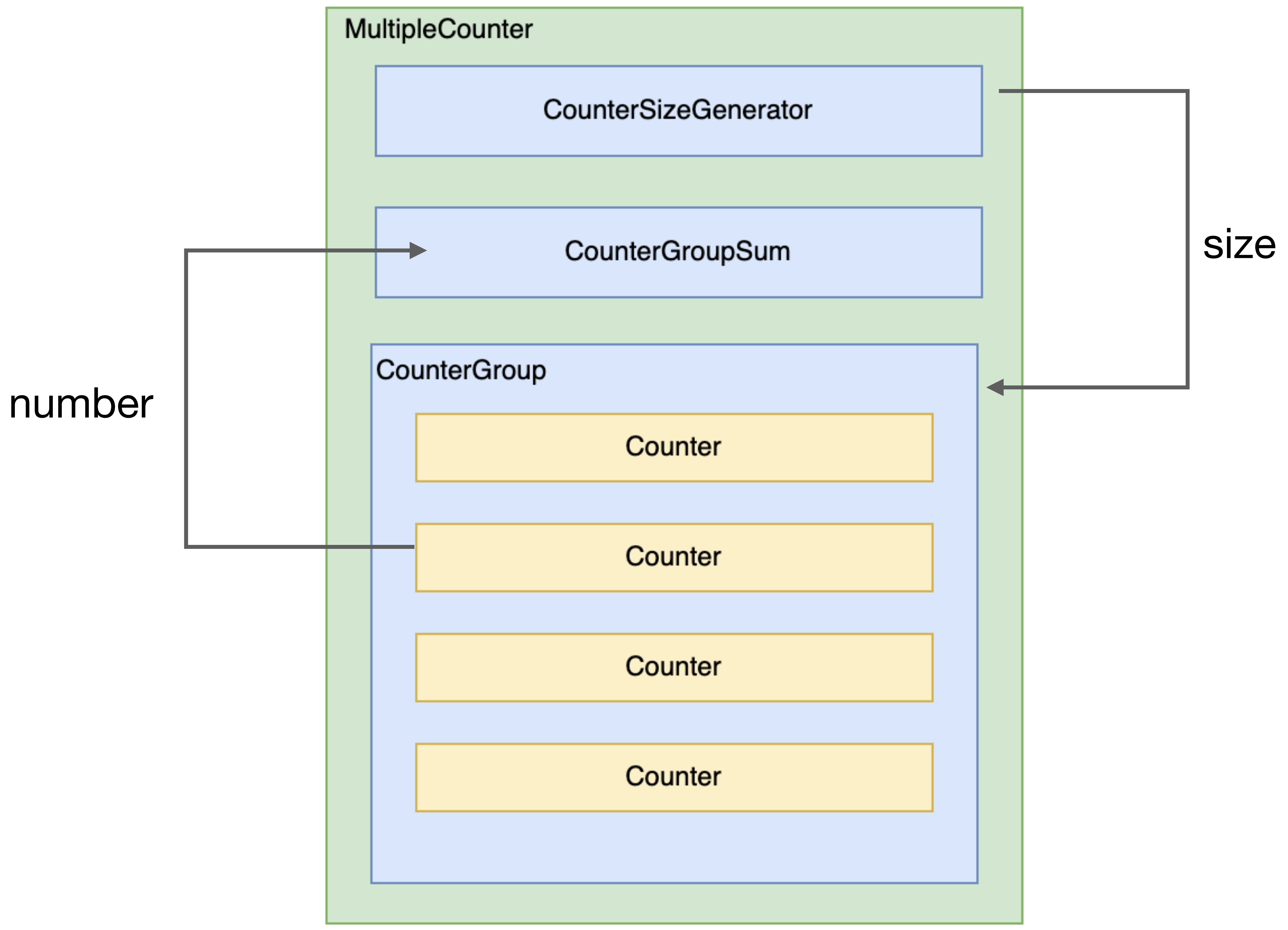
Counter Practice



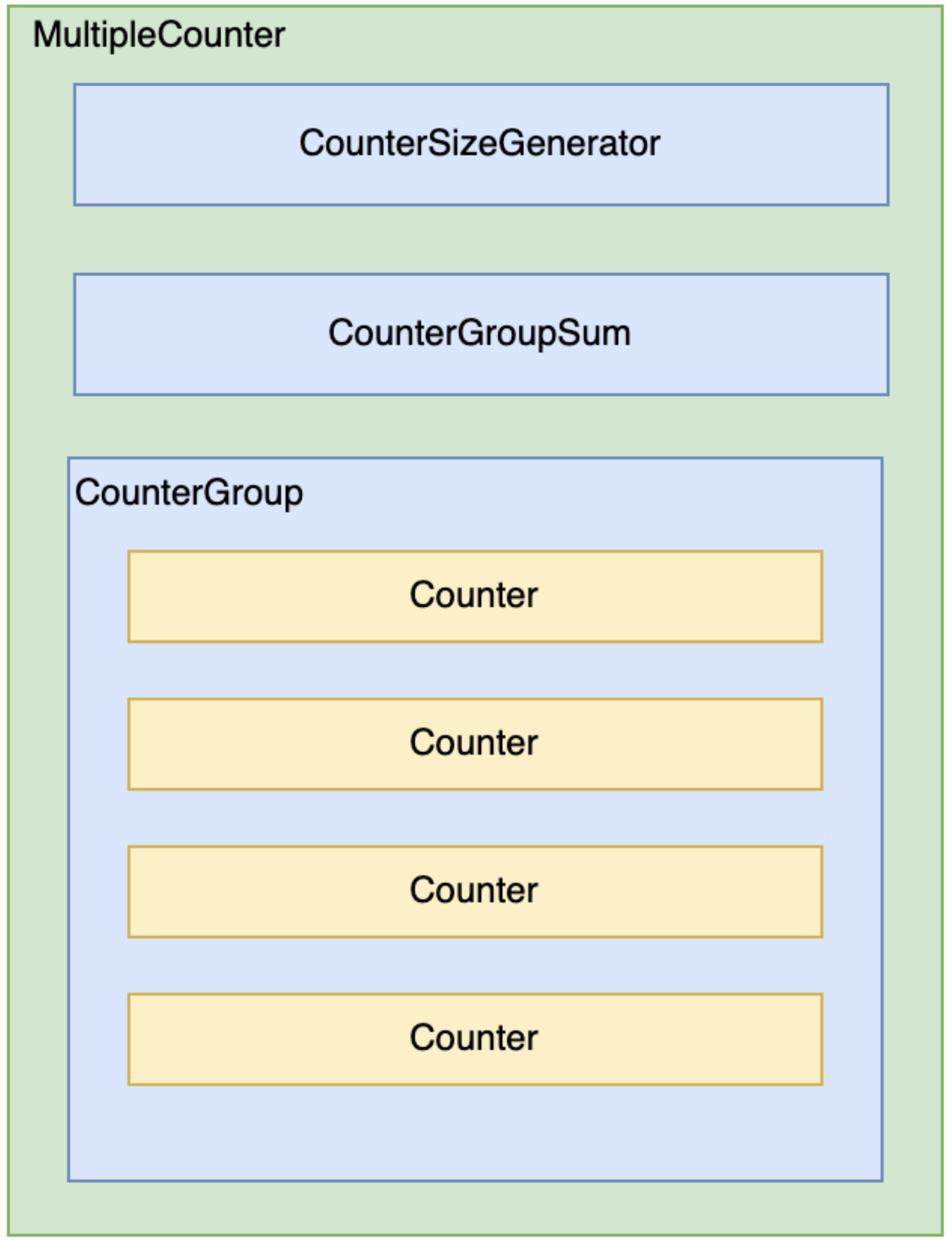


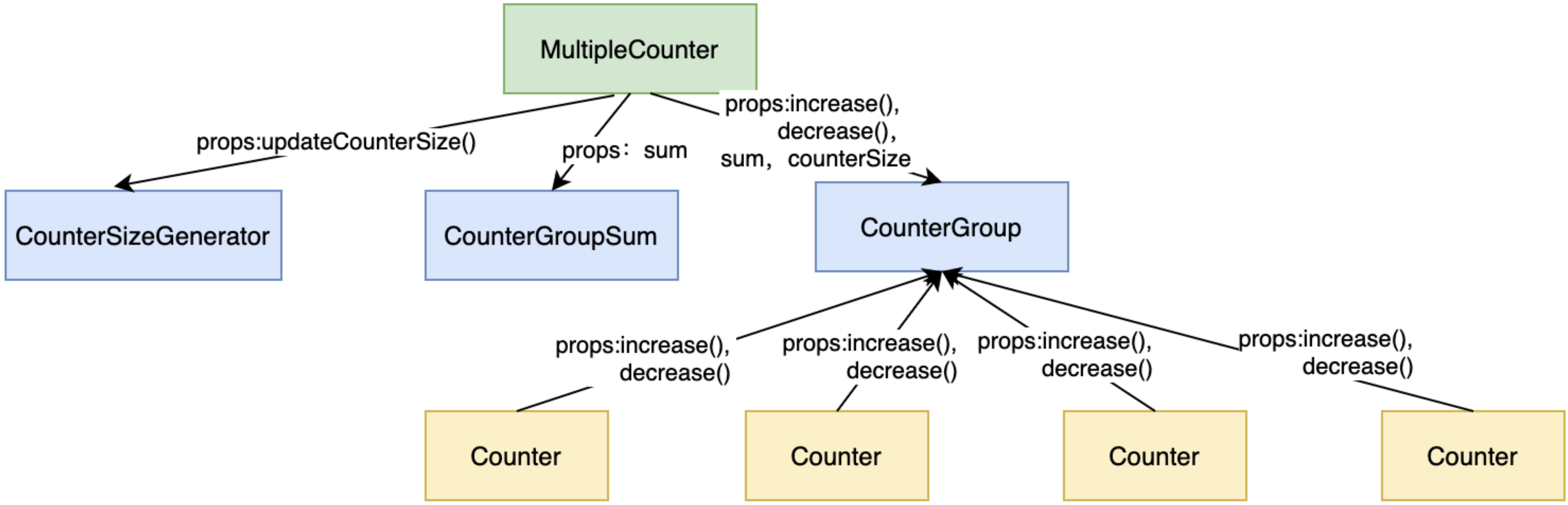


Multiple Counter

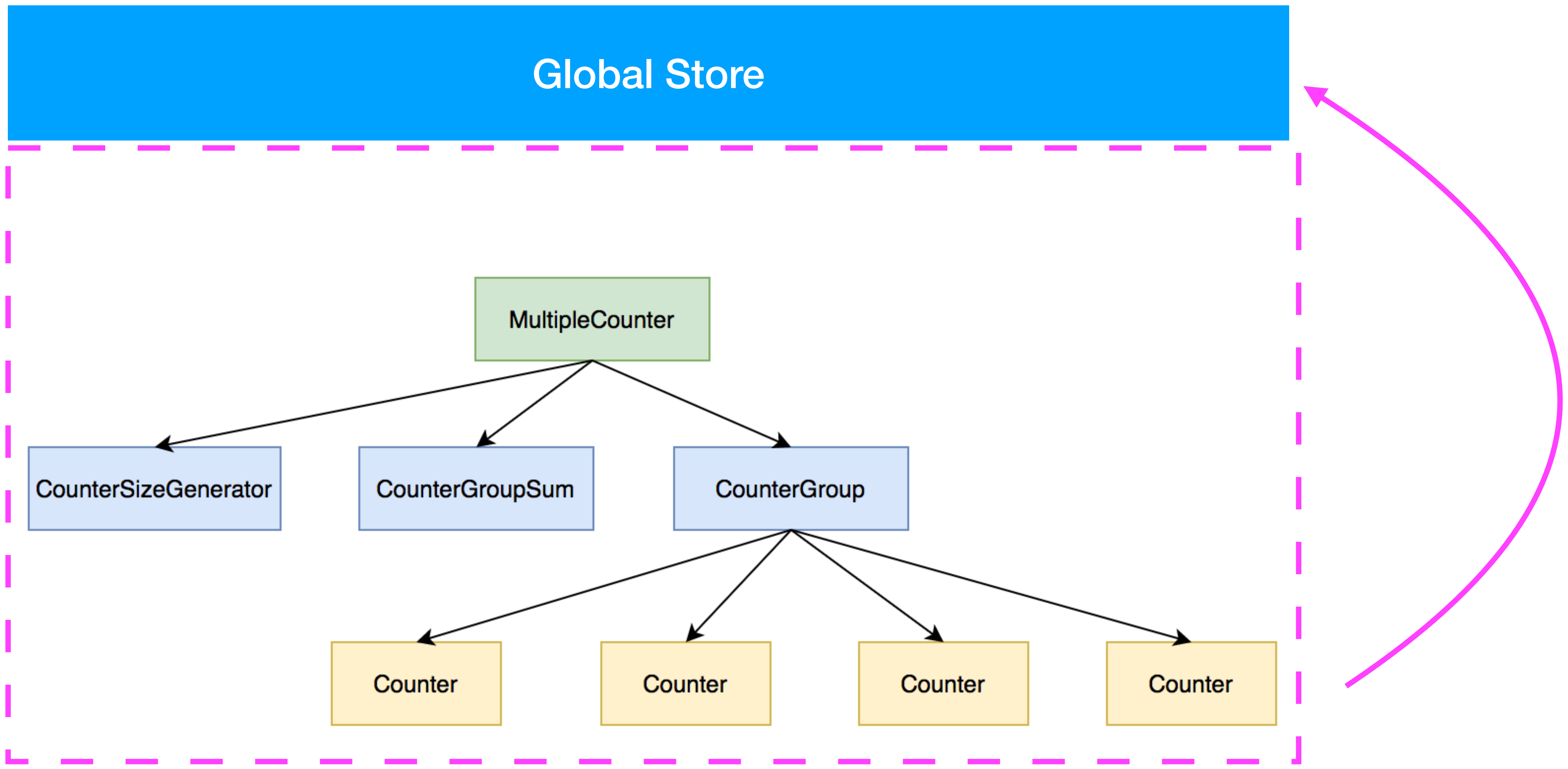


Multiple Counter

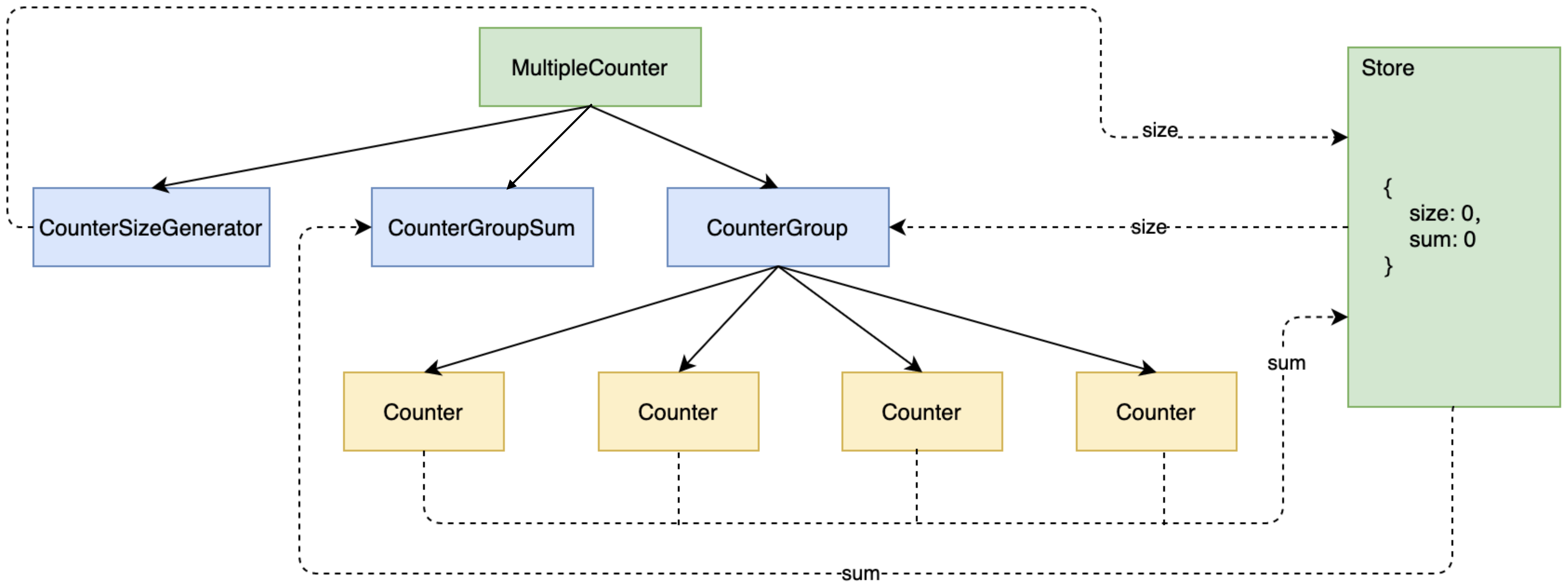



MultipleCounter
props:updateCounterSize()
props： sum
props:increase(), decrease(), sum， counterSize
CounterSizeGenerator
CounterGroupSum
CounterGroup
props:increase(), decrease()
props:increase(), decrease()
props:increase(), decrease()
props:increase(), decrease()
Counter
Counter
Counter
Counter

Multiple Counter

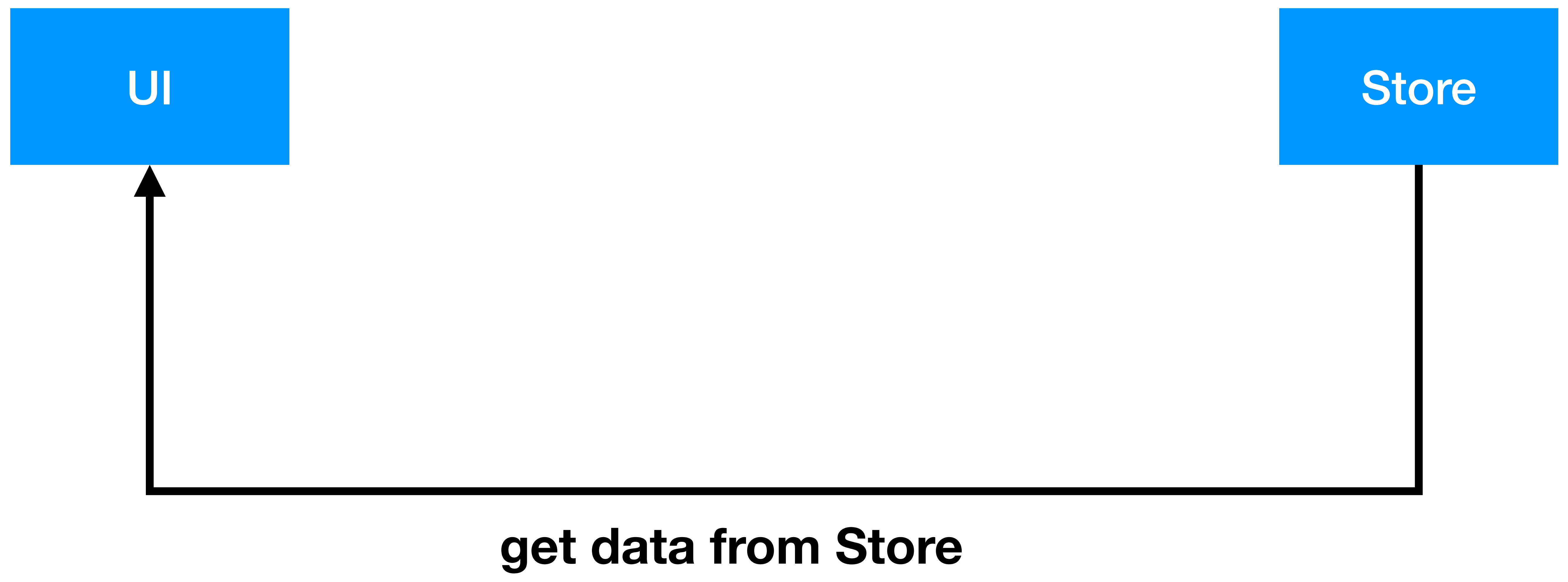


Multiple Counter

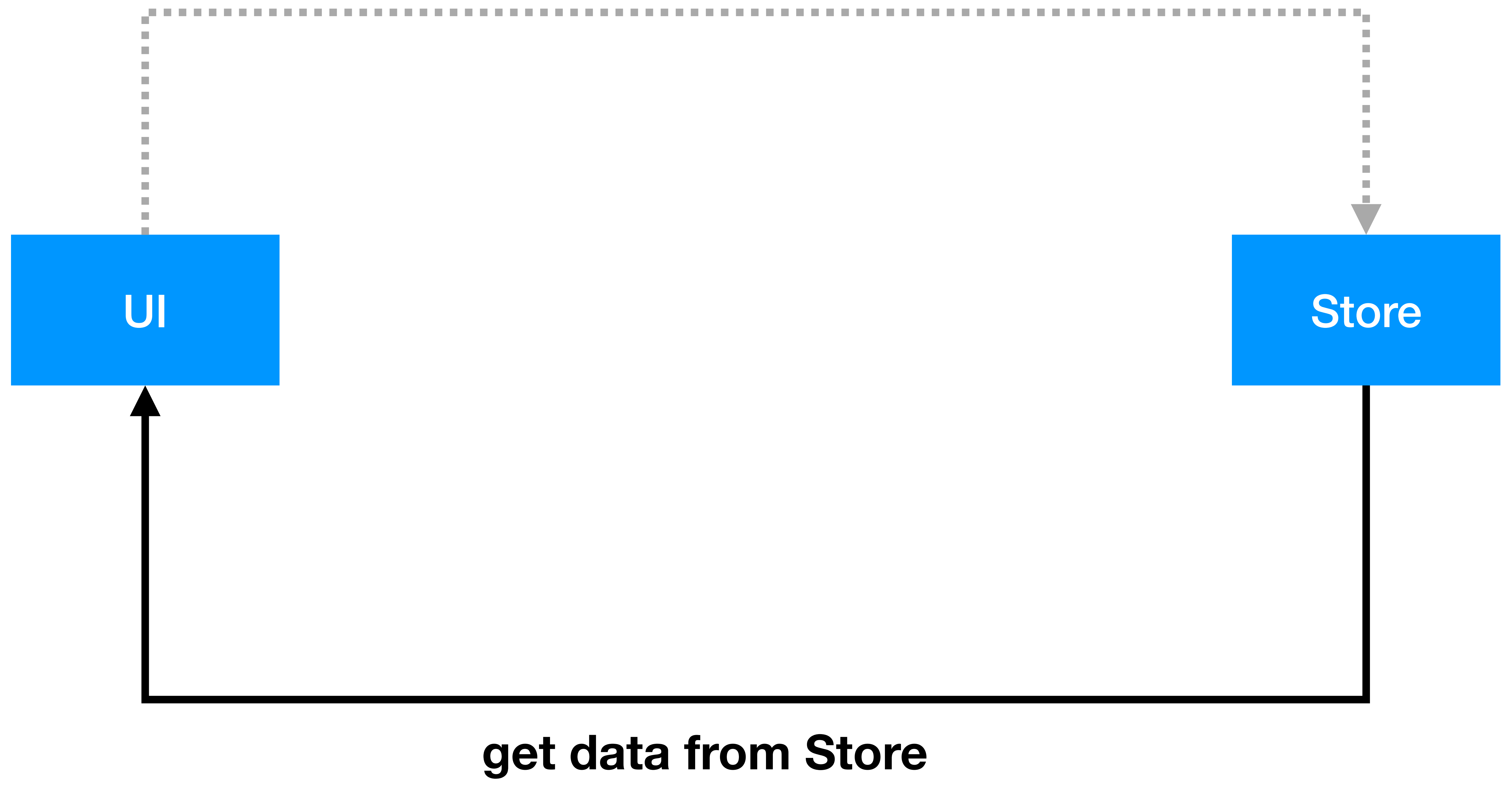


Components relationship

Components relationship



Components relationship

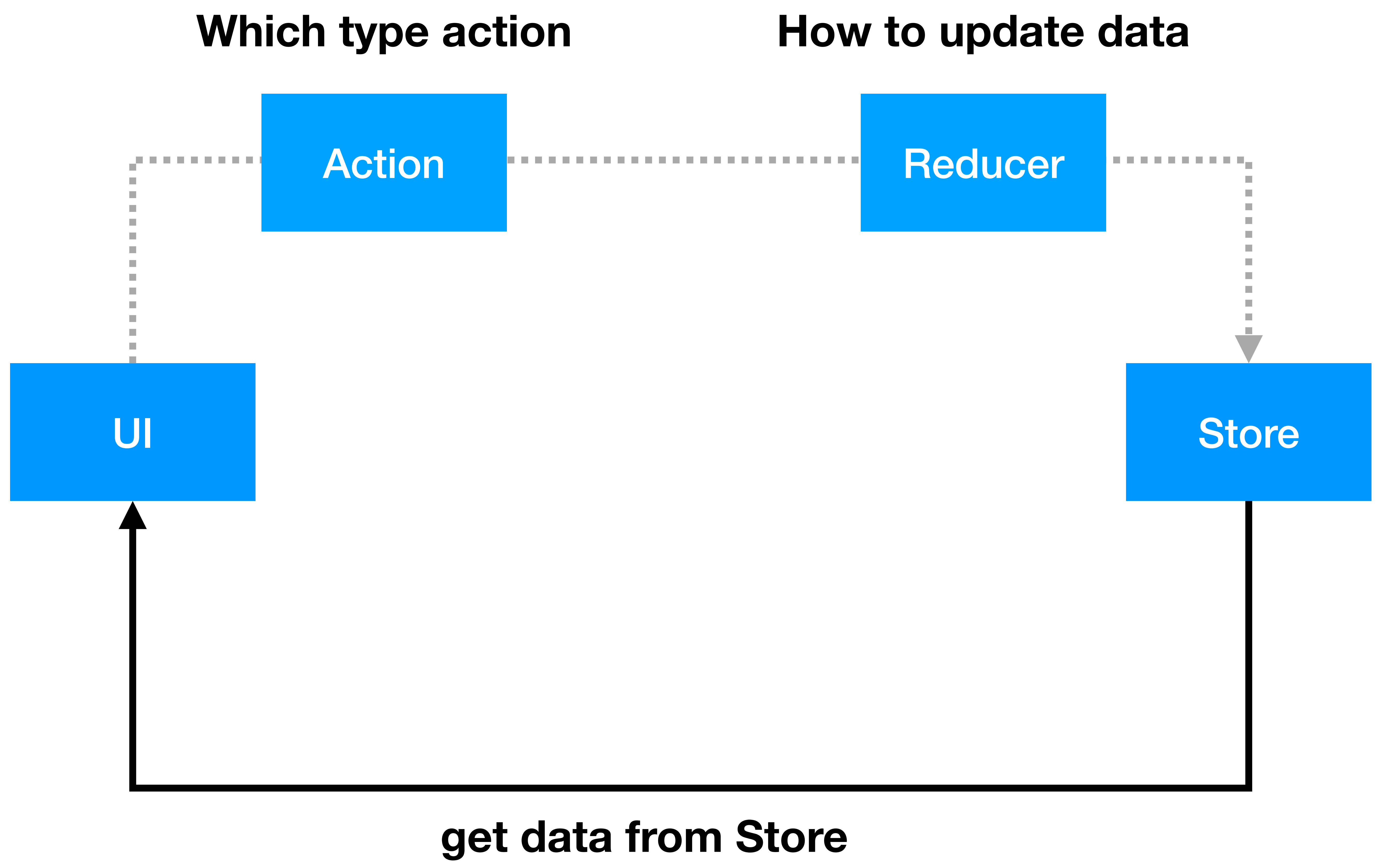


Redux

Redux is a pattern and library for managing and updating **"global" state**.

It serves as a **centralized store** for state that needs to be used across many parts of your application.

Redux


Which type action
How to update data
Action
Reducer
UI
Store
get data from Store

React redux

Redux itself is a standalone library that can be used with any UI layer or framework, including React, Angular, Vue, Ember …

We need to use **react-redux** to **bind React and Redux** to let you use store management in react component.

Use React redux

1, Install **redux and react-redux** to use React Redux with your React app:

npm install @reduxjs/toolkit react-redux

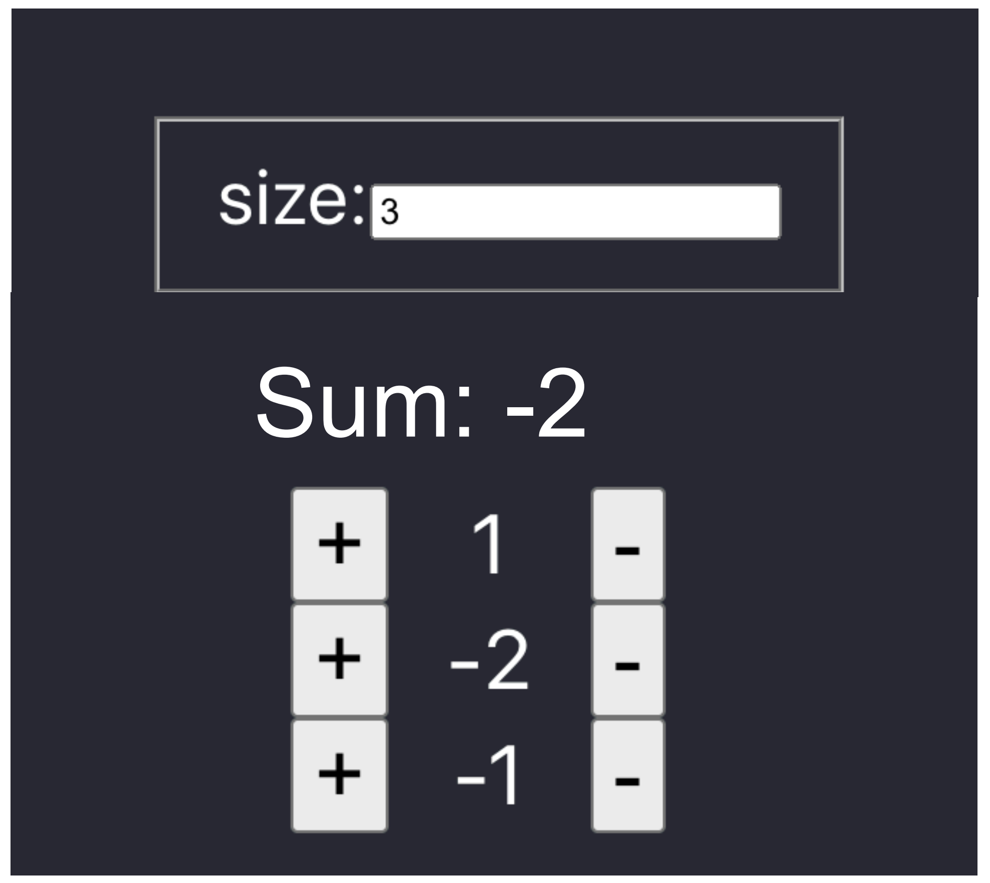
Based on the project created using: npx create-react-app my-app

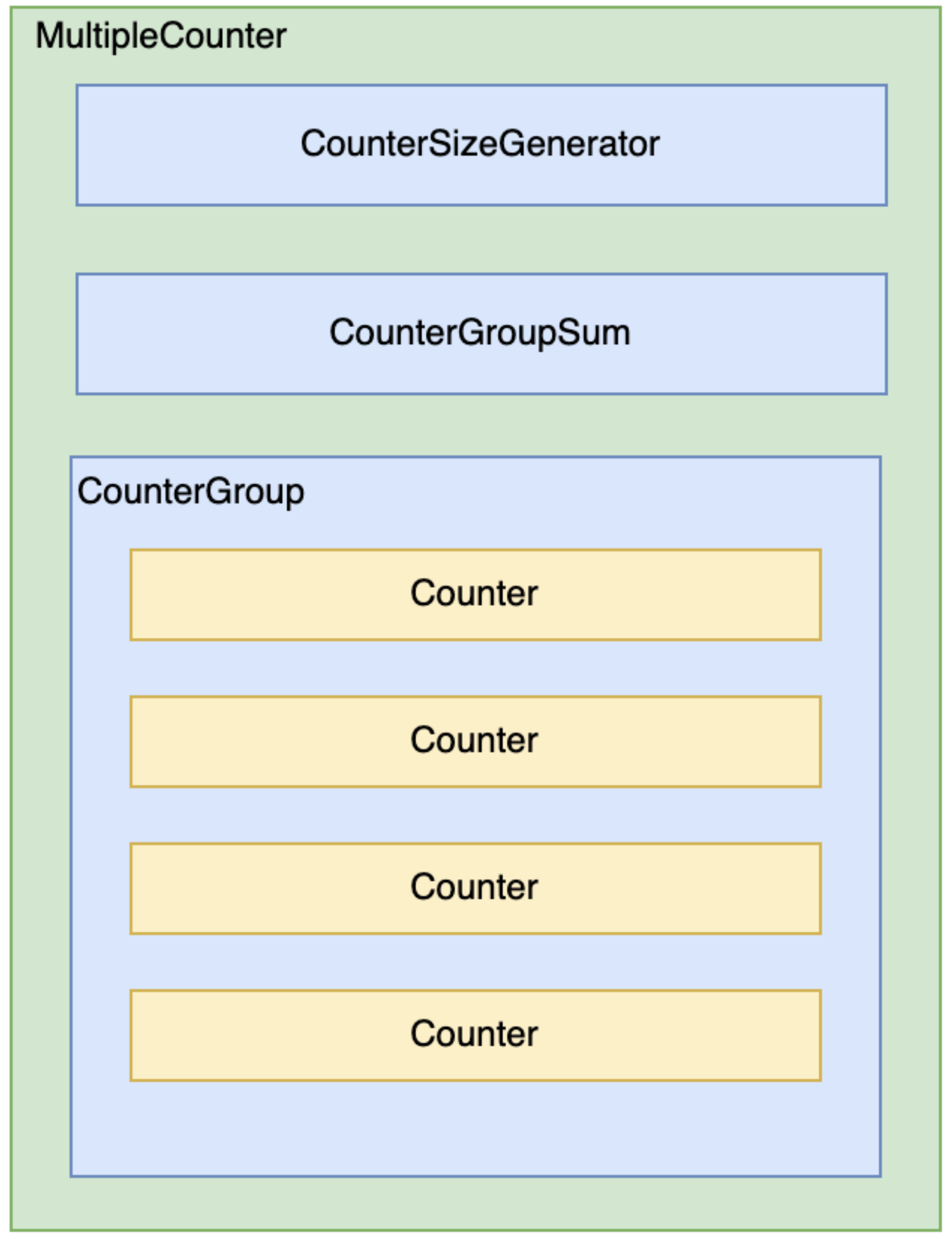
Reference: https://redux.js.org/tutorials/quick-start

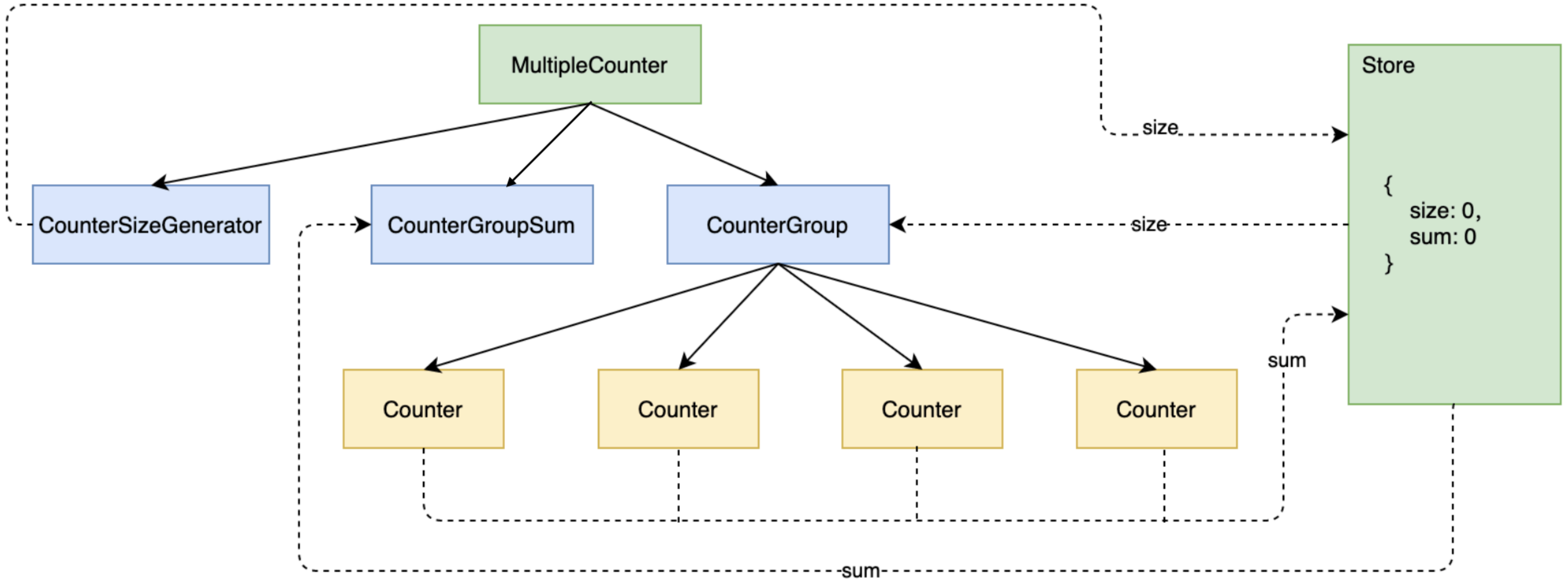
2, Add extensions in chrome:

- React DevTools Extension:
  - React DevTools Extension for Chrome
  - React DevTools Extension for Firefox
- Redux DevTools Extension:
  - Redux DevTools Extension for Chrome
  - Redux DevTools Extension for Firefox

Counter Practice

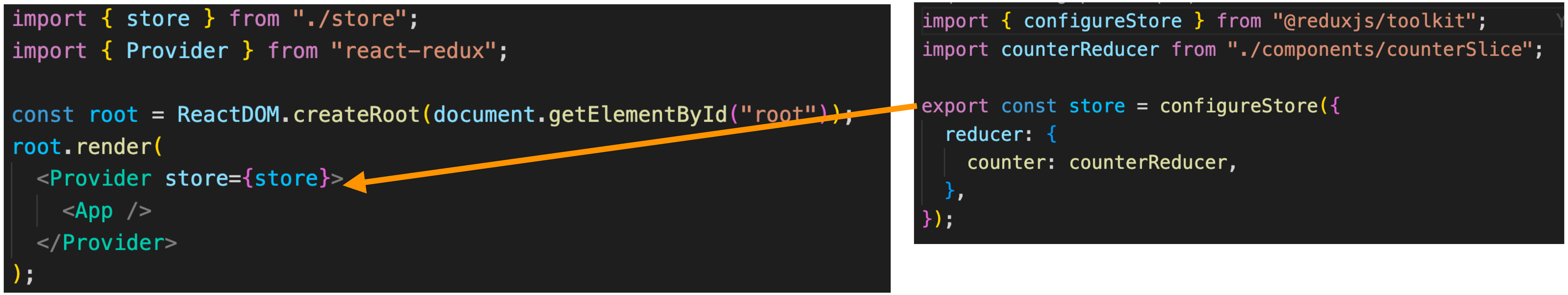






Provider (from “react-redux")

React Redux provides <Provider />, which makes the Redux store available to your app:

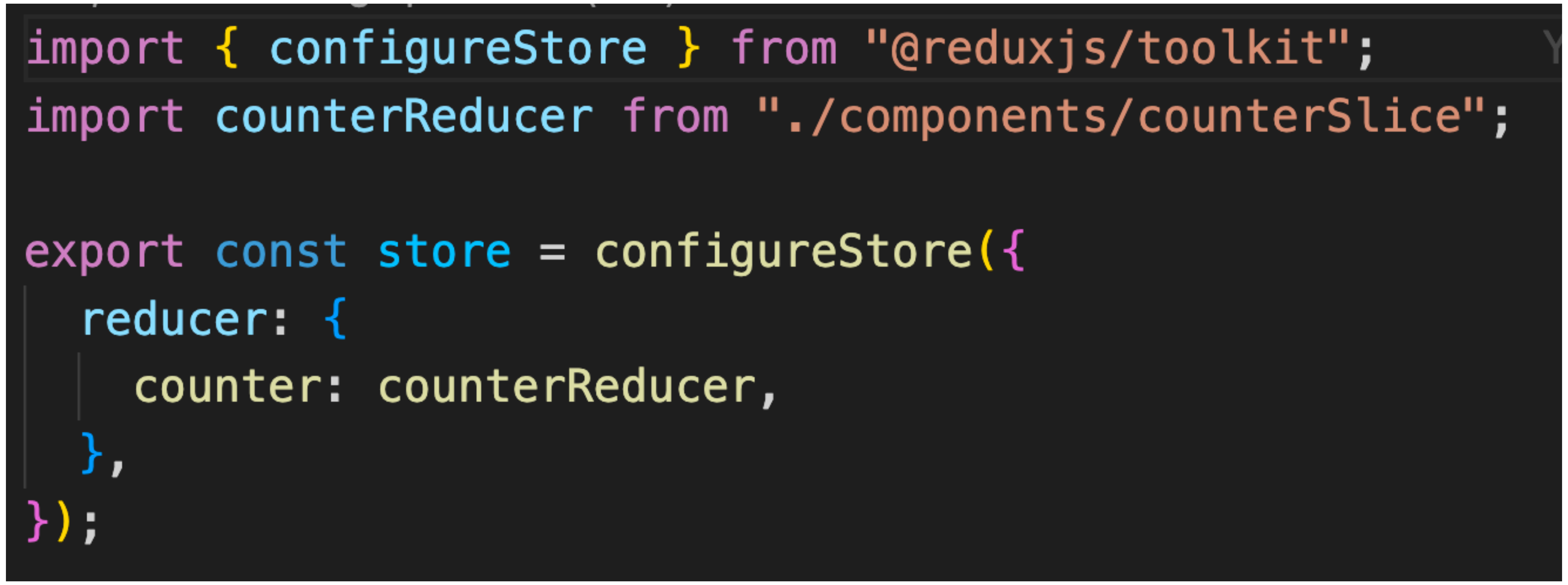


```
import { store } from "./store";
import { Provider } from "react-redux";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

```
import { configureStore } from "@reduxjs/toolkit";
import counterReducer from "./components/counterSlice";

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
});
```

ConfigureStore (from “@reduxjs/toolkit")

Store is the object that brings all the application state together.

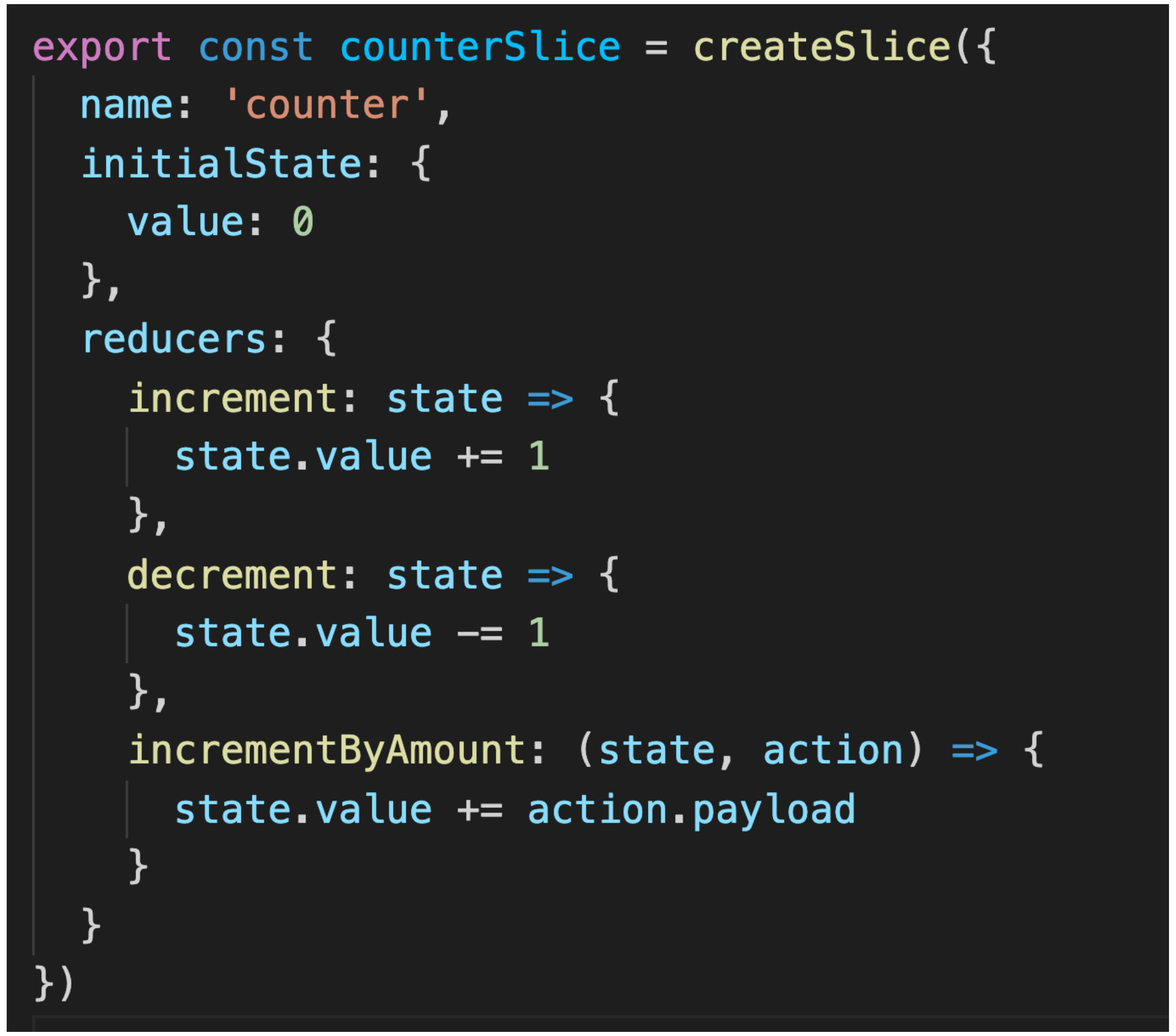

```
import { configureStore } from "@reduxjs/toolkit";
import counterReducer from "./components/counterSlice";

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
});
```

Redux state is typically organized into "slices", defined by the reducers.

CreateSlice (from “@reduxjs/toolkit”)

Redux Toolkit includes a createSlice function that will auto-generate the action types and action creators for you, based on the names of the reducer functions you provide.

Reducers specify **how the application's state changes** in response to actions sent to the store.



```
export const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0
  },
  reducers: {
    increment: state => {
      state.value += 1
    },
    decrement: state => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    }
  }
})
```

```
Actions:
{type: "counter/increment"}
{type: "counter/decrement"}
{type: "counter/incrementByAmount"}
```

https://redux-toolkit.js.org/api/createSlice

https://redux-toolkit.js.org/usage/usage-guide#creating-slices-of-state

TodoList Practice

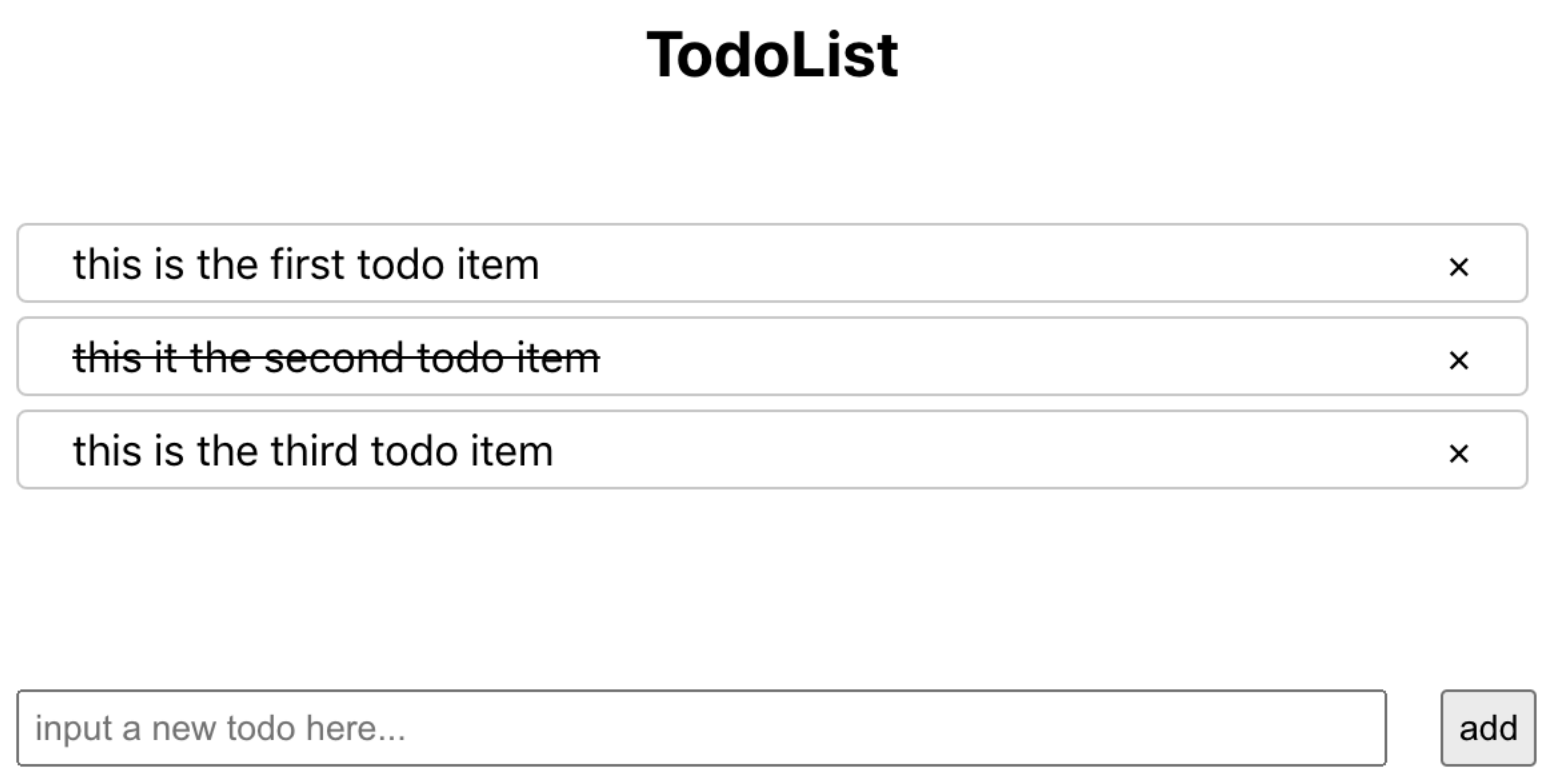
Implement a todo list:

1. I can add a todo item, the item will present in the todo list when added.

2. When I click a todo item, the todo item will be marked as done with line through, and the item status will change to done.

3. When I click the done item, the line through will be cancelled, and the item status will revert to undone.

3. I can delete a todo item when I click the cross stamp.

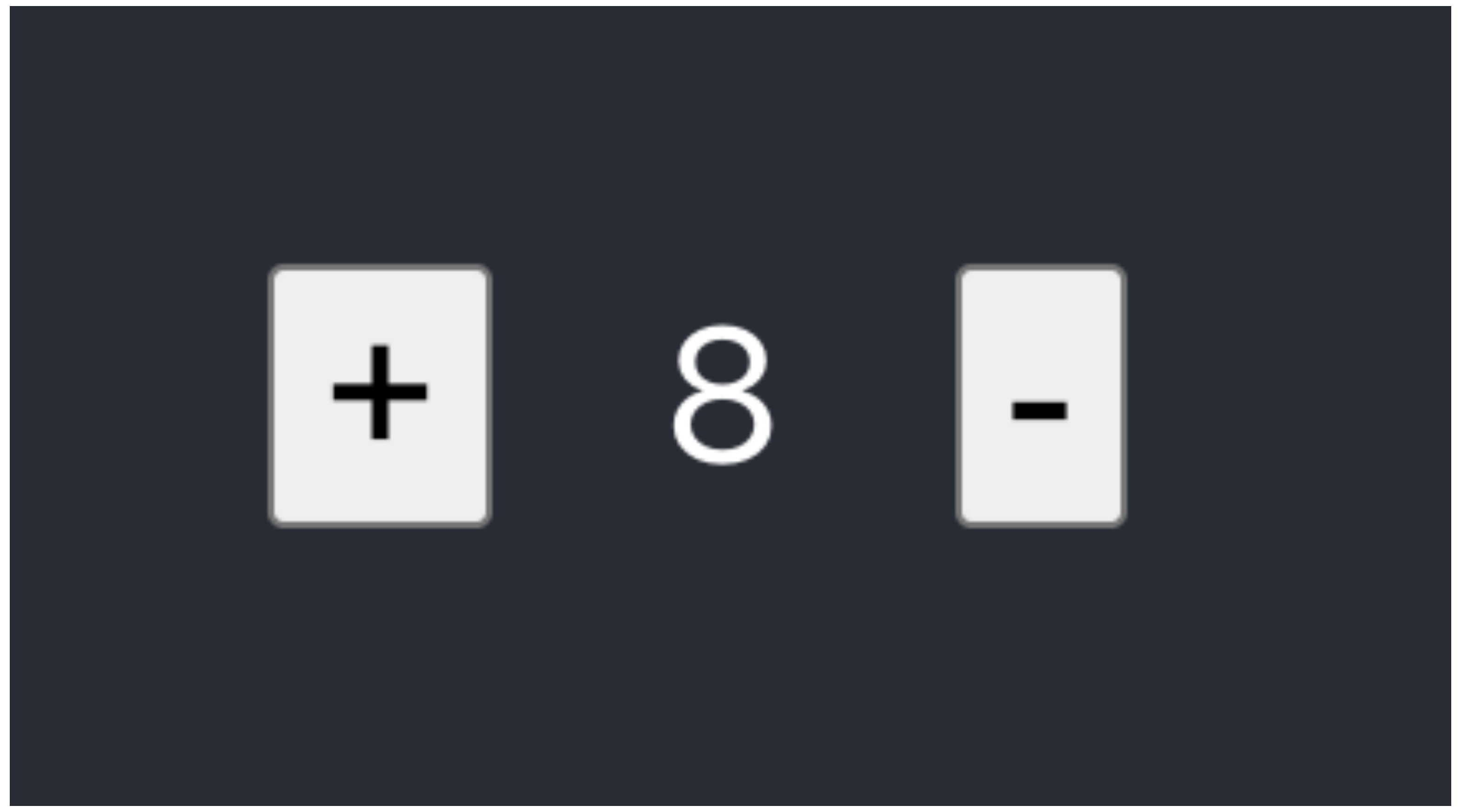

TodoList

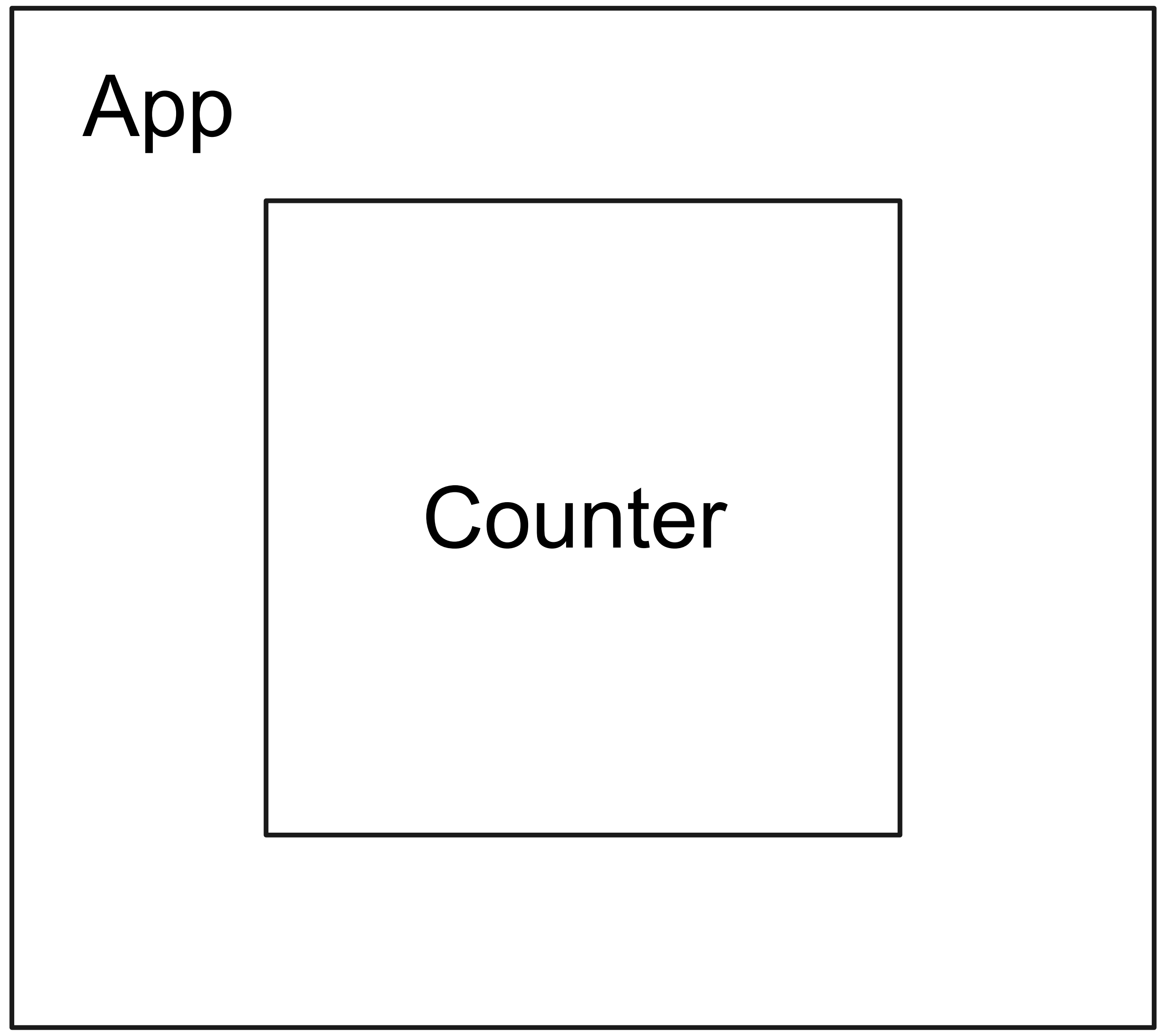
this is the first todo item ×

~~this it the second todo item~~ ×

this is the third todo item ×

Practice - Counter



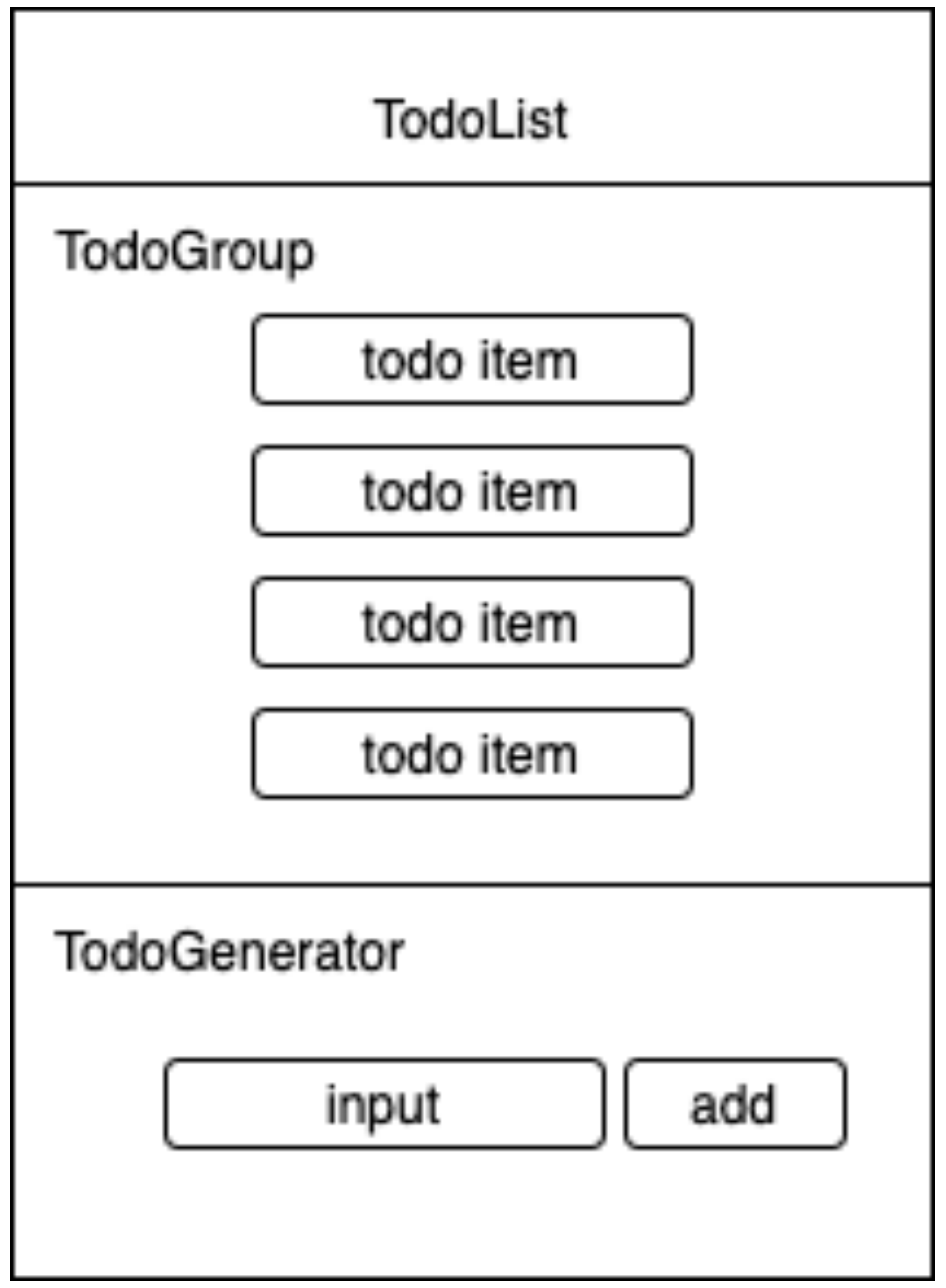


Pair Programming

1	Michael	Jenny
2	Thomas	Alvin
3	Alan	Marie
	Heinrich	

4	Antony	Vincent
5	Joyce	Chris
6	Polly	Kelvin

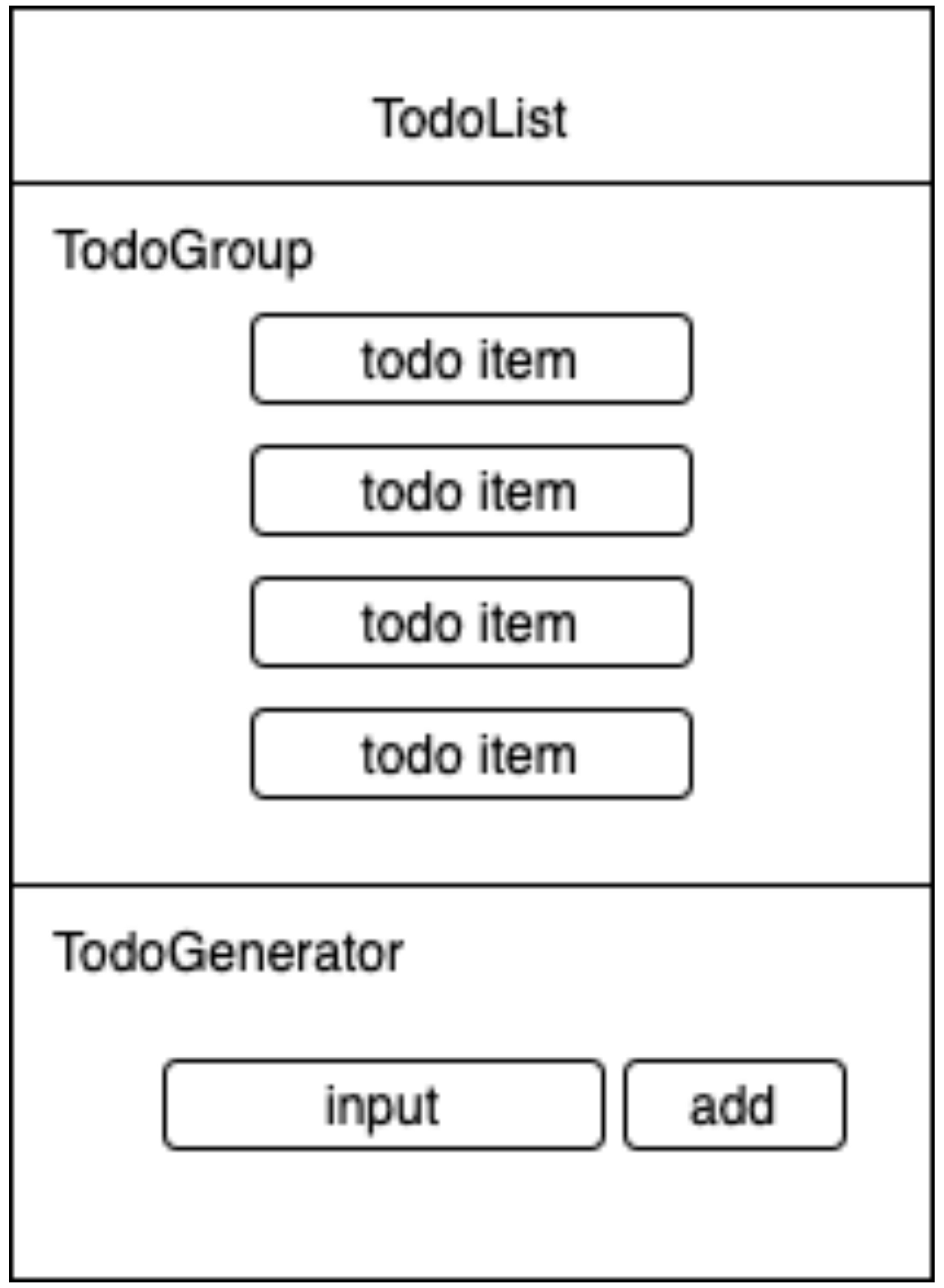
TodoList Components



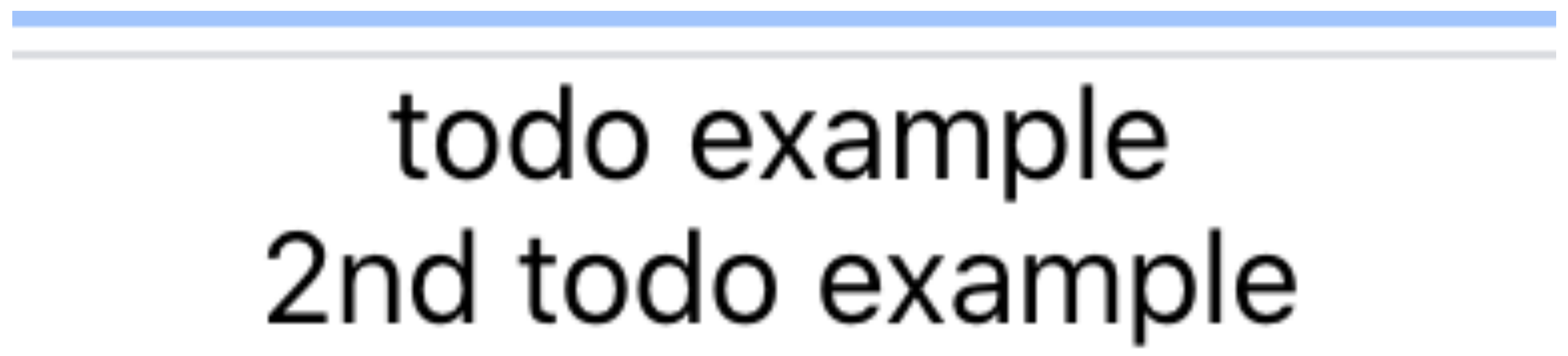
todoList:

```
[
  {
    id: "cc53dc26-61b0-406b-99dd-b8825dd2ceec"
    text: "todo example"
    done: false
  },
  {
    id: "dd53dc26-b061-6b40-dd99-82b85dd2ce90"
    text: “first todo item"
    done: false
  }
]
```

TodoList Components

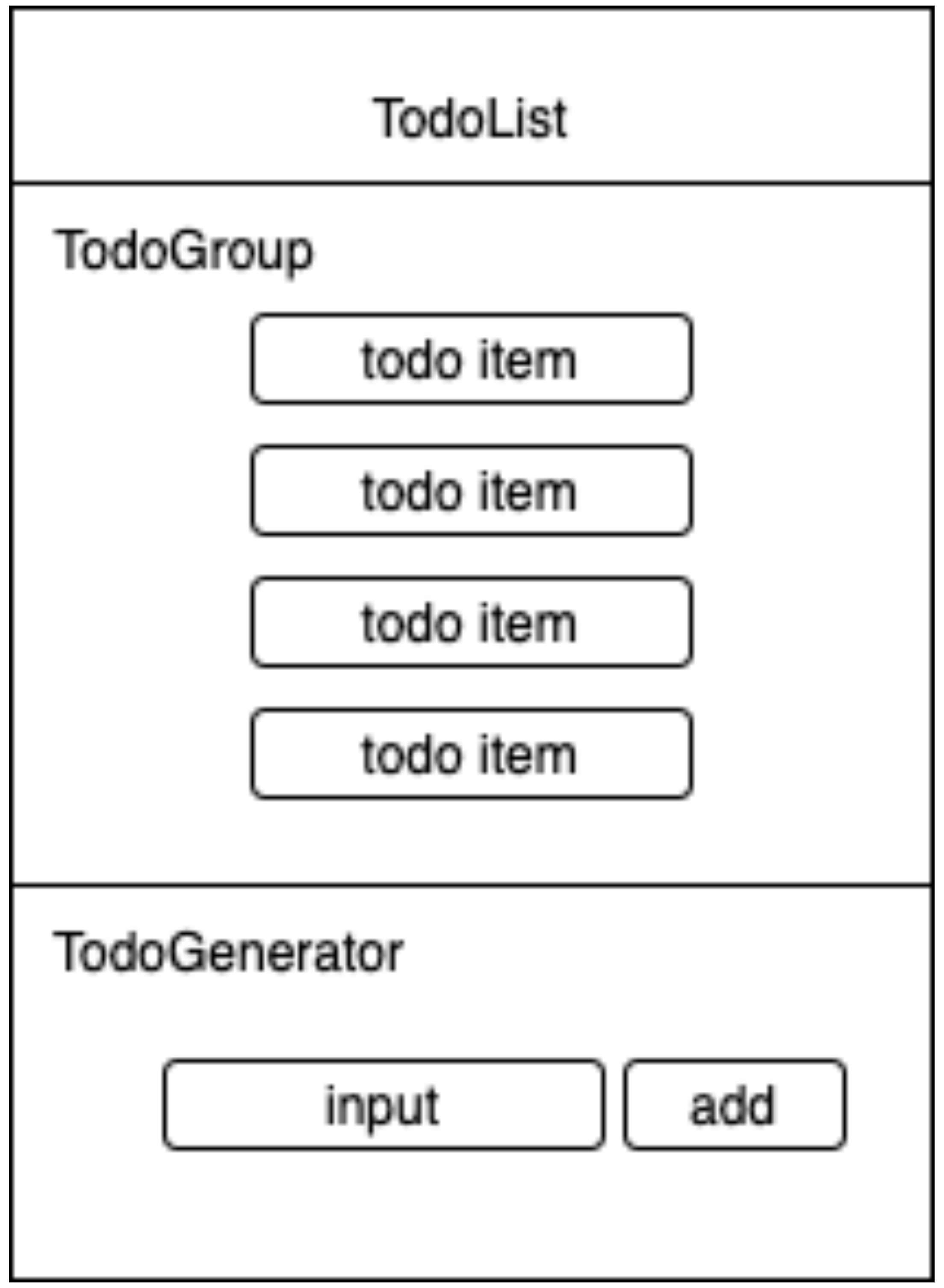


1. **Create a todo group to show initial state**
   - Static Components with static data
     - TodoGroup component
     - TodoItem component


todo example
2nd todo example

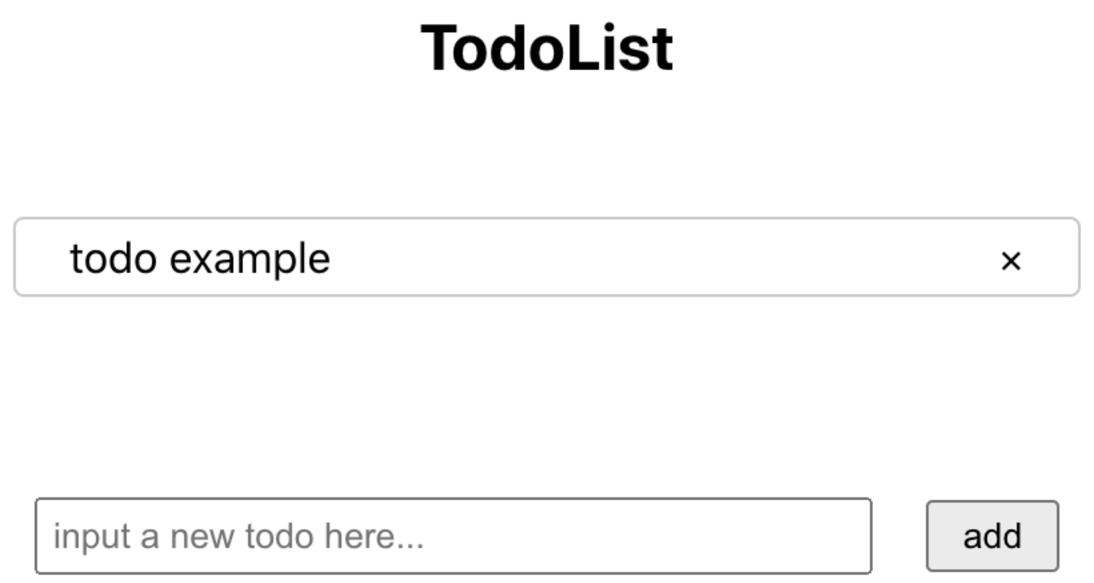
```
[
  {
    id: "cc53dc26-61b0-406b-99dd-b8825dd2ceec"
    text: "todo example"
    done: false
  }
]
```

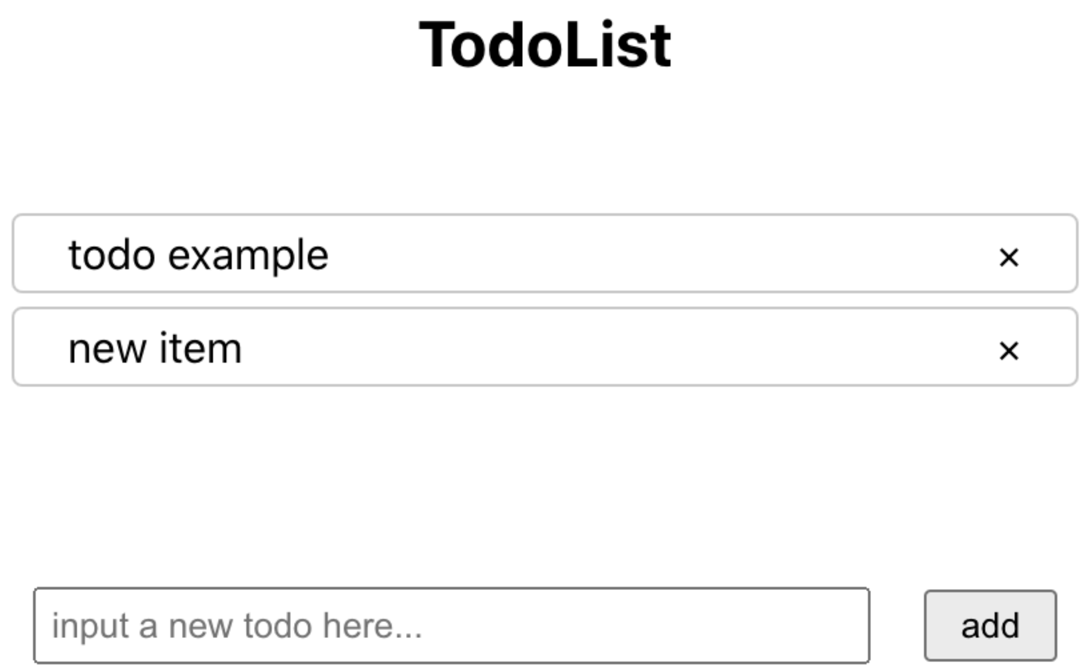
TodoList Components



1. Create a todo list to show initial state

2. Create a add todo component

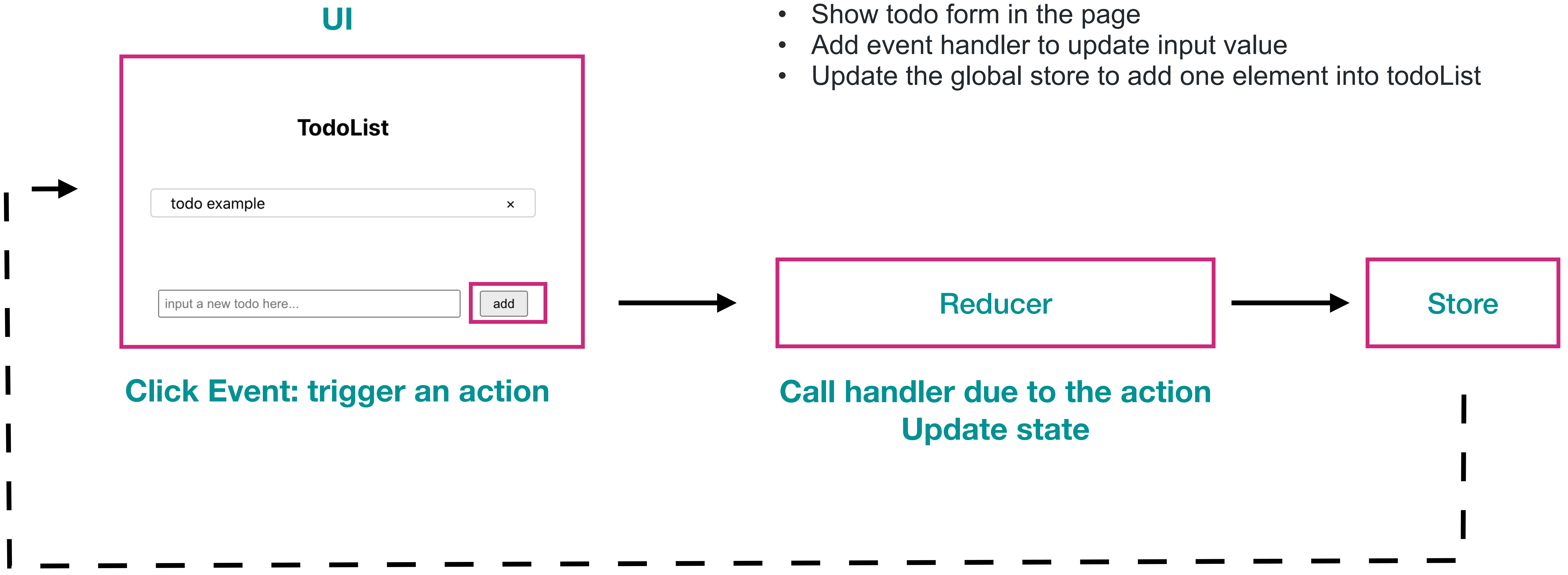




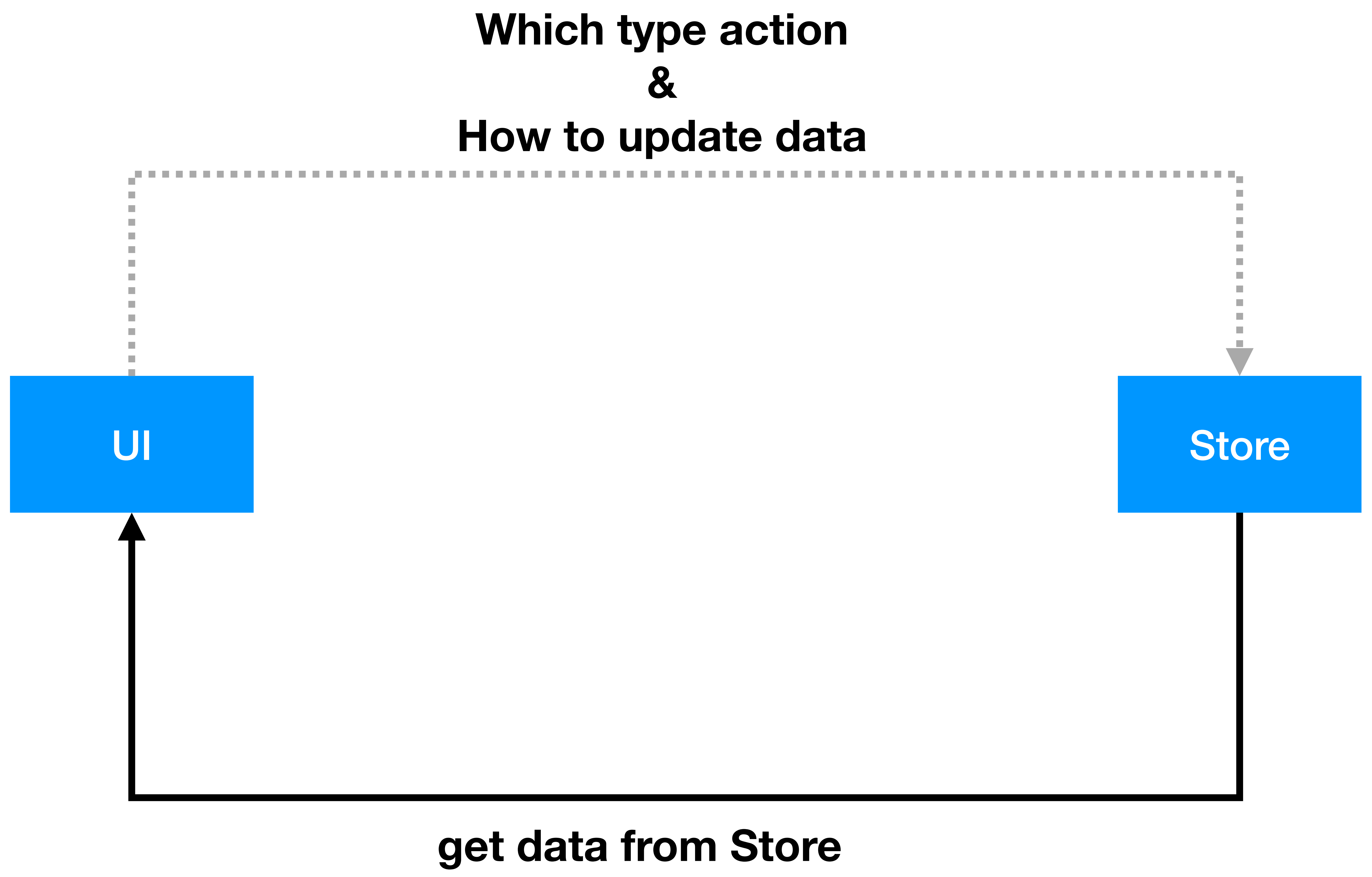
TodoList Components

1. Create a todo list to show initial state

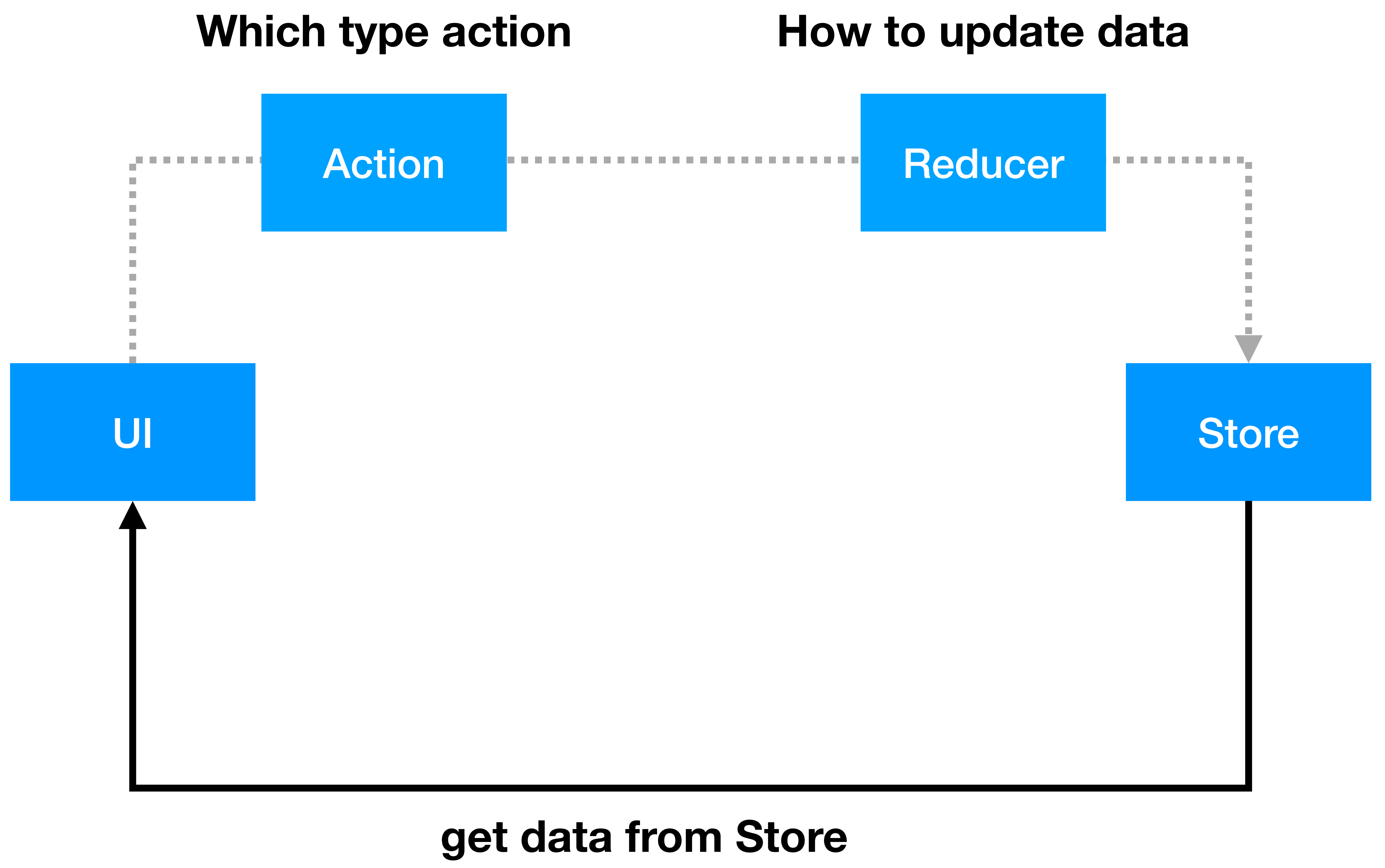
2. **Create a add todo component**
   - Show todo form in the page
   - Add event handler to update input value
   - Update the global store to add one element into todoList



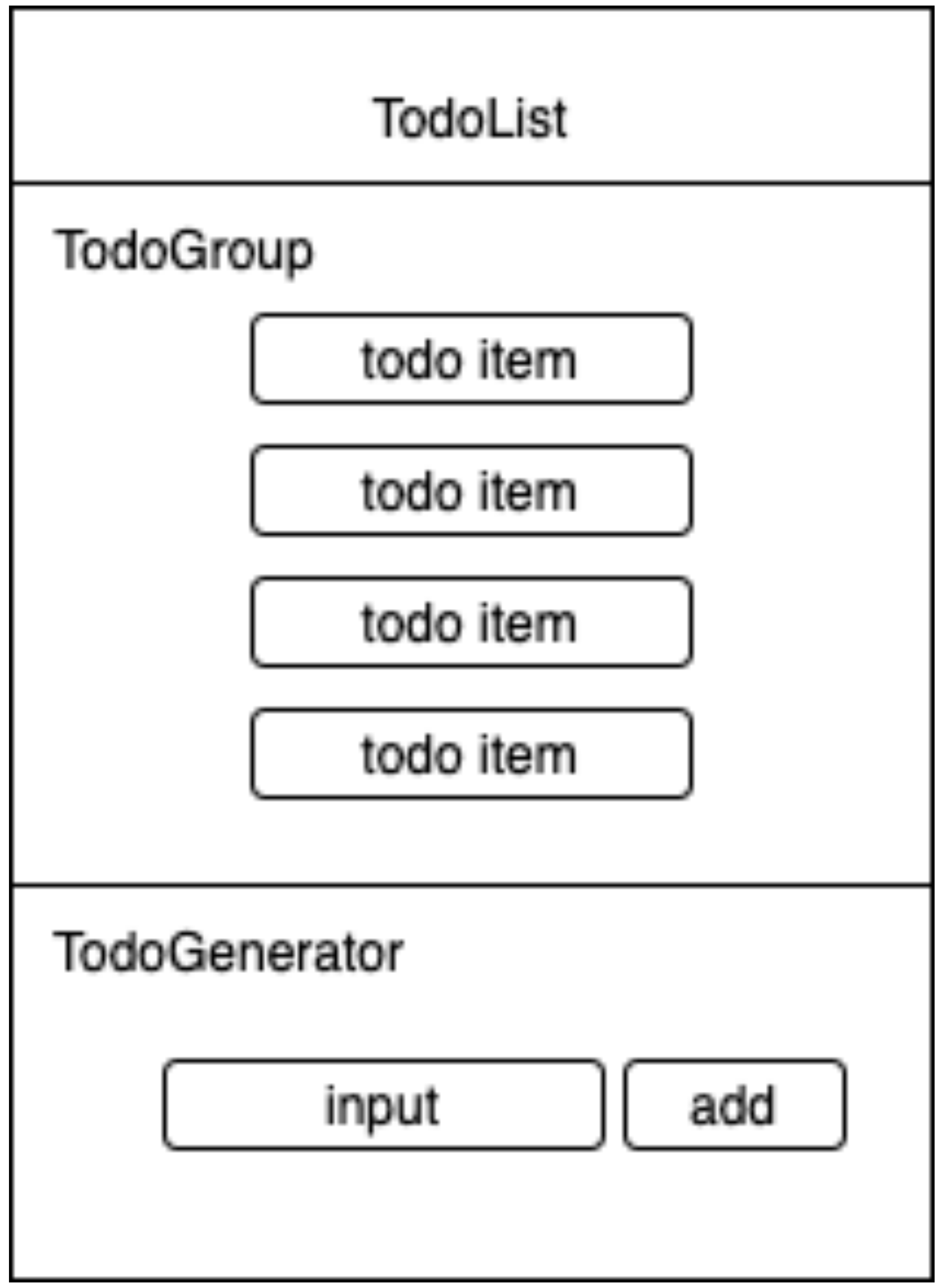
Components relationship



Components relationship



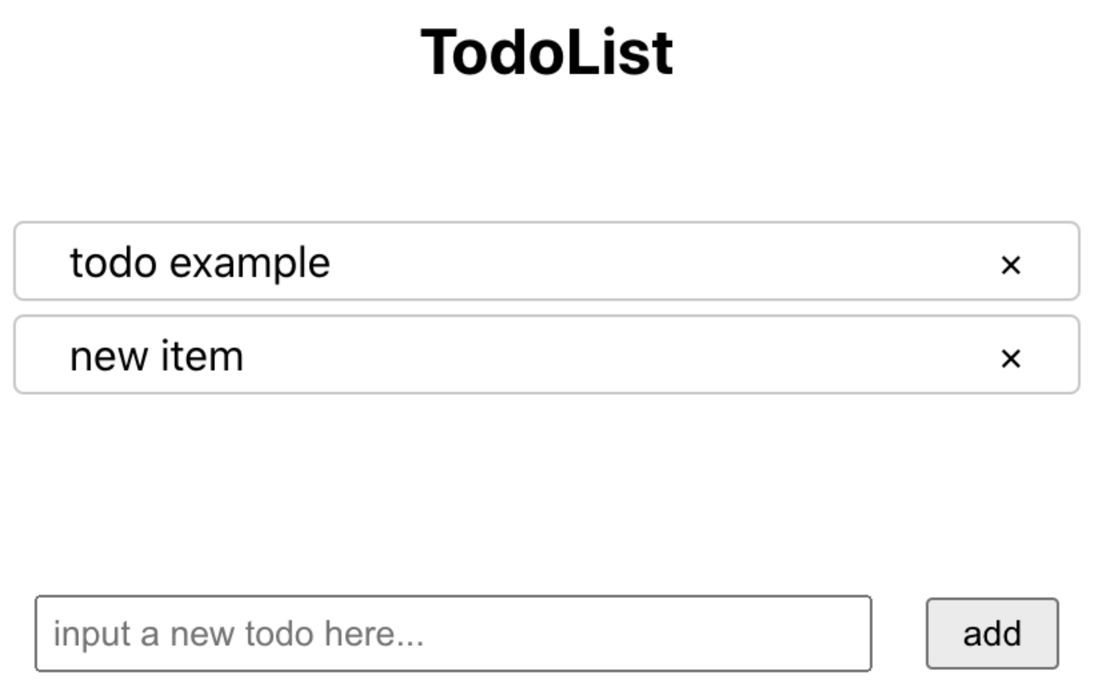
TodoList Components



1. Create a todo list to show initial state

2. Create a add todo component

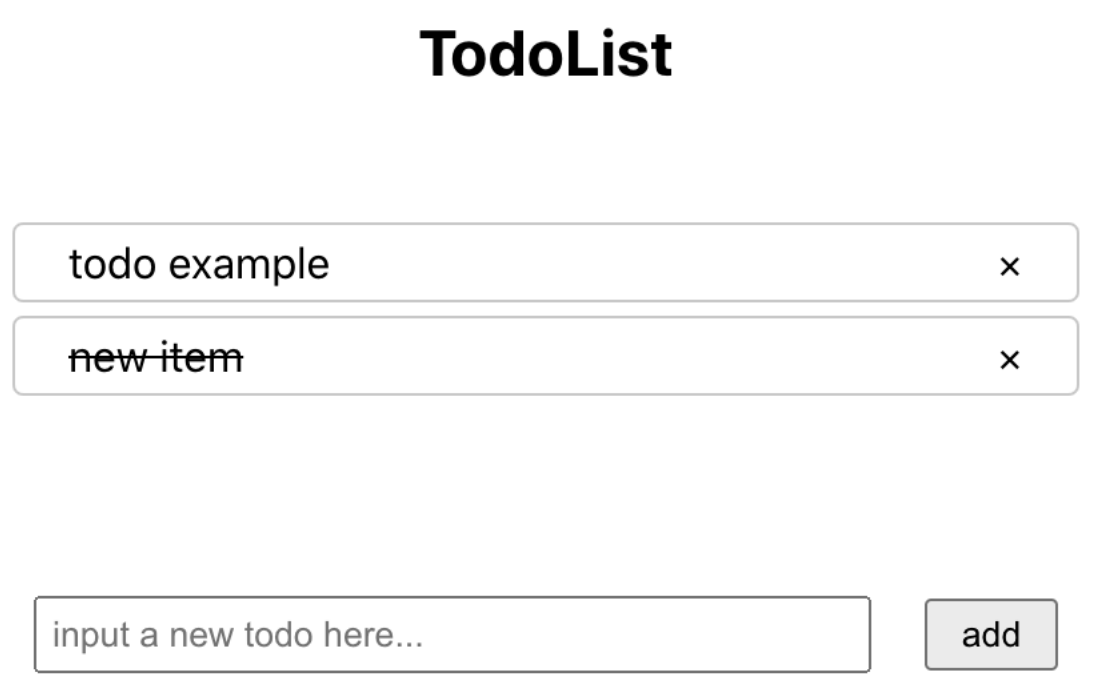
3. **Toggle a todo item**
   - Add event handler when click one todo item
   - Update the global store to update the clicked item status
   - Add style for the completed todo item



TodoList

todo example ×

new item ×

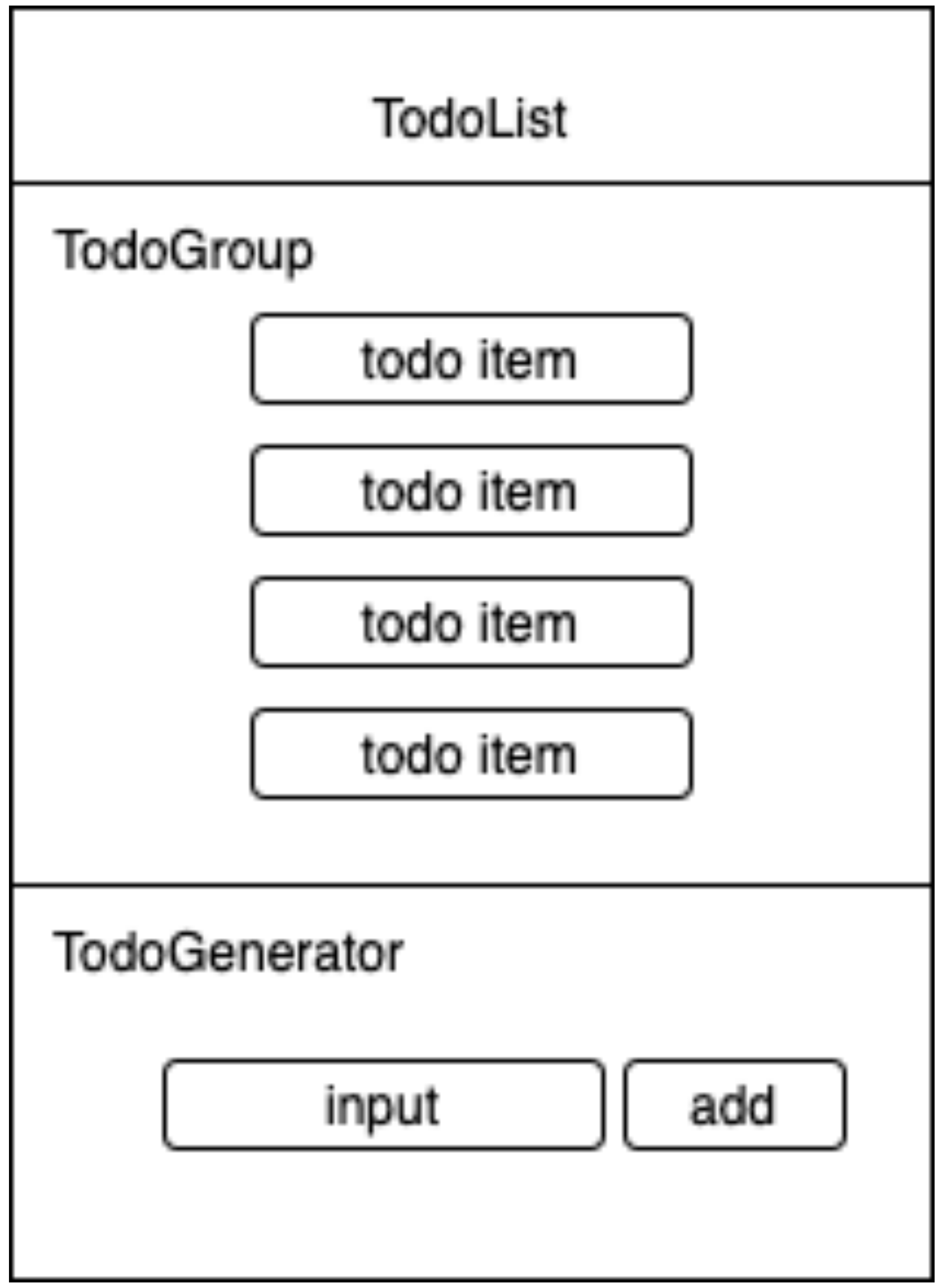


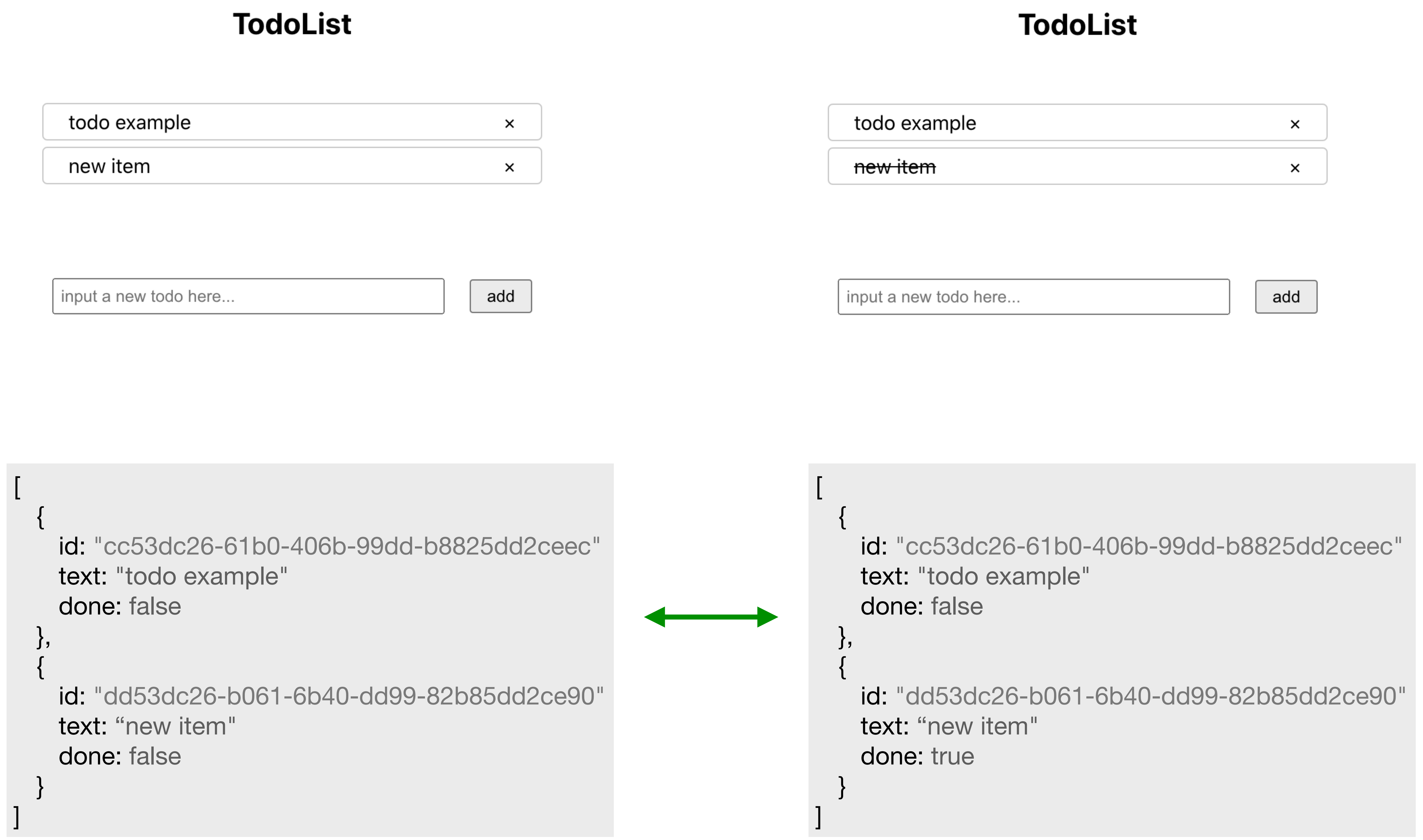
TodoList

todo example ×

~~new item~~ ×

TodoList Components

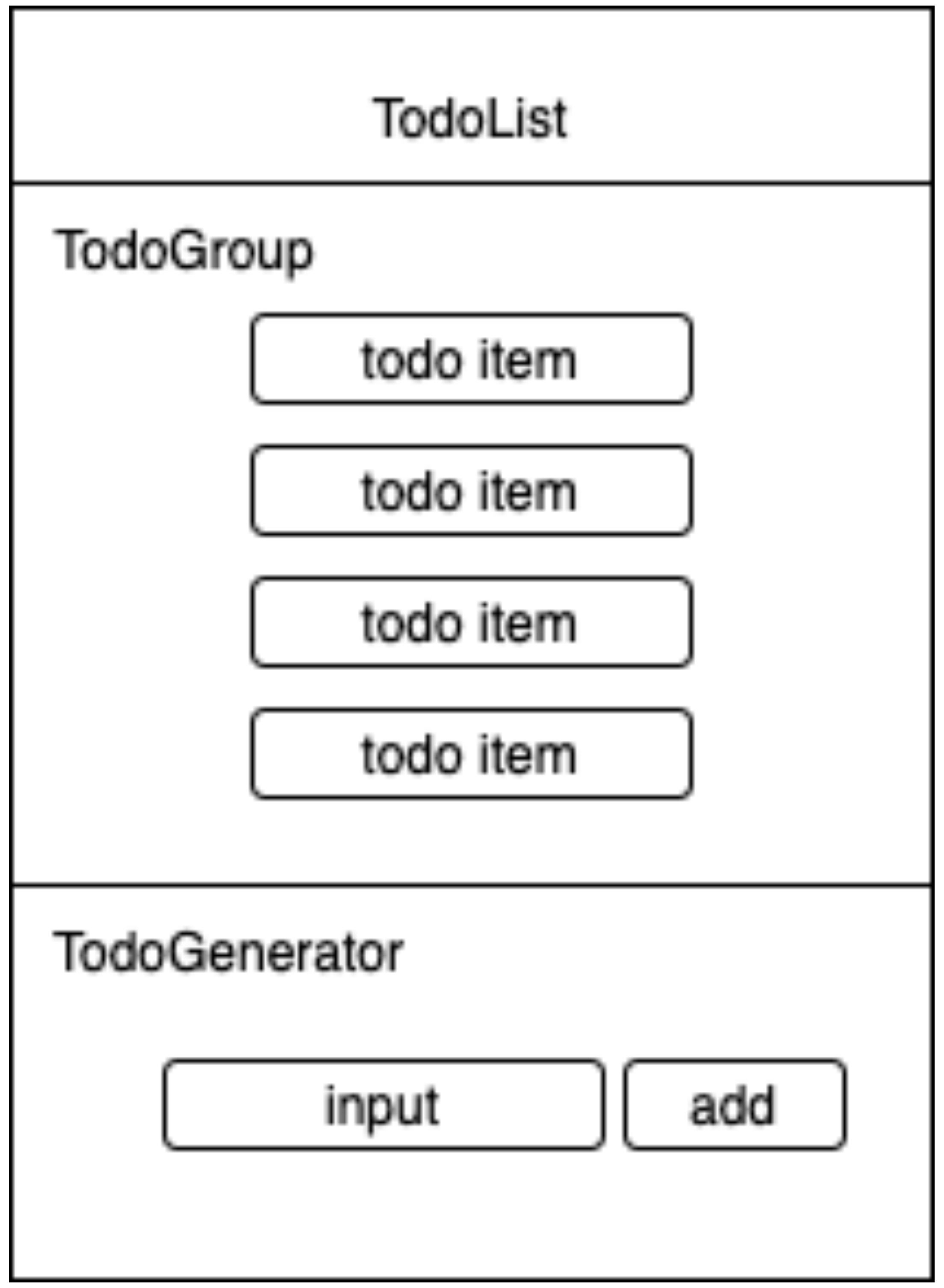




```
[
  {
    id: "cc53dc26-61b0-406b-99dd-b8825dd2ceec"
    text: "todo example"
    done: false
  },
  {
    id: "dd53dc26-b061-6b40-dd99-82b85dd2ce90"
    text: “new item"
    done: false
  }
]
```

```
[
  {
    id: "cc53dc26-61b0-406b-99dd-b8825dd2ceec"
    text: "todo example"
    done: false
  },
  {
    id: "dd53dc26-b061-6b40-dd99-82b85dd2ce90"
    text: “new item"
    done: true
  }
]
```

TodoList Components

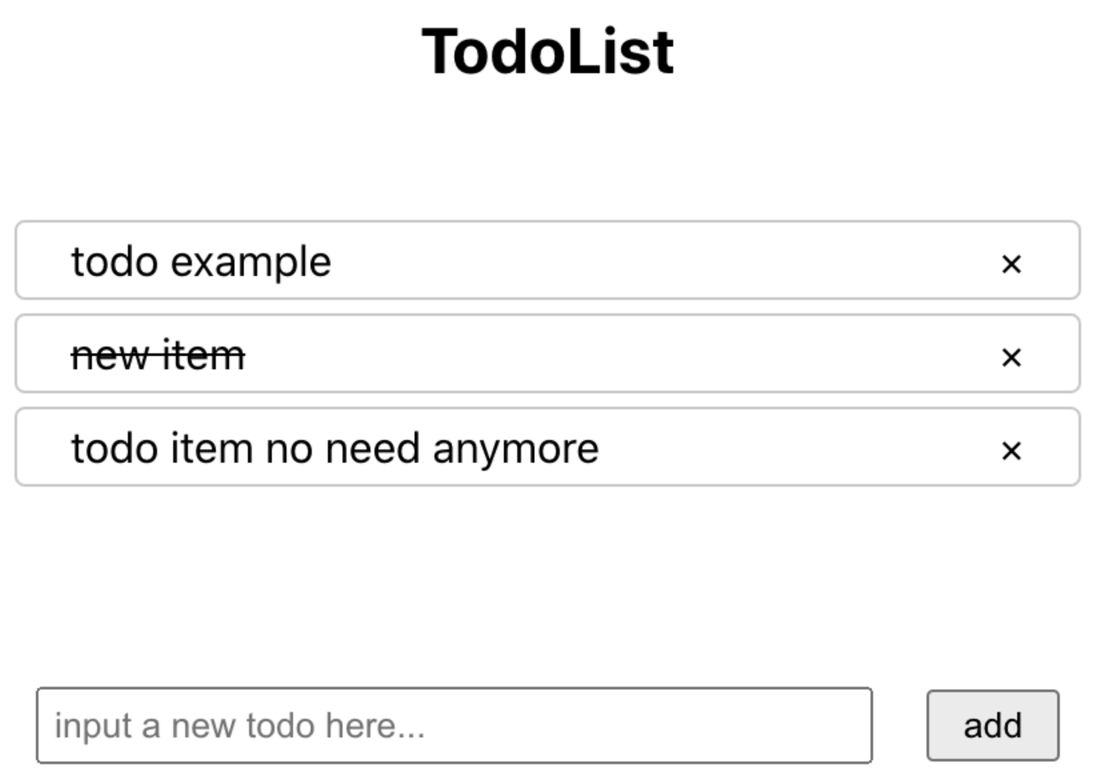


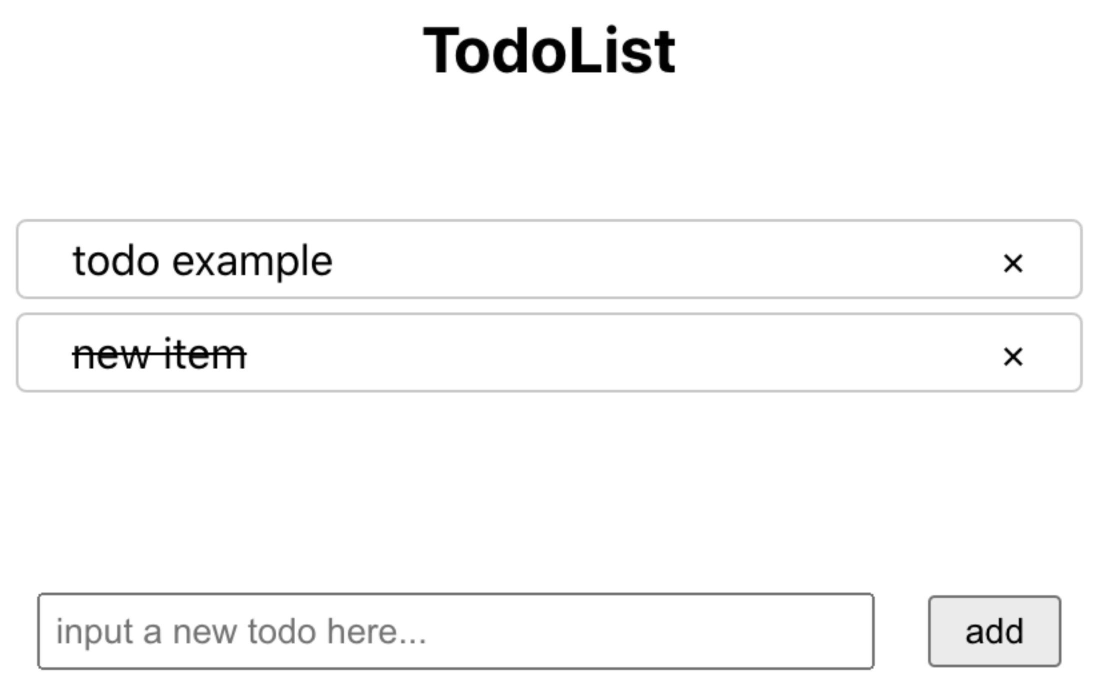
1. Create a todo list to show initial state

2. Create a add todo component

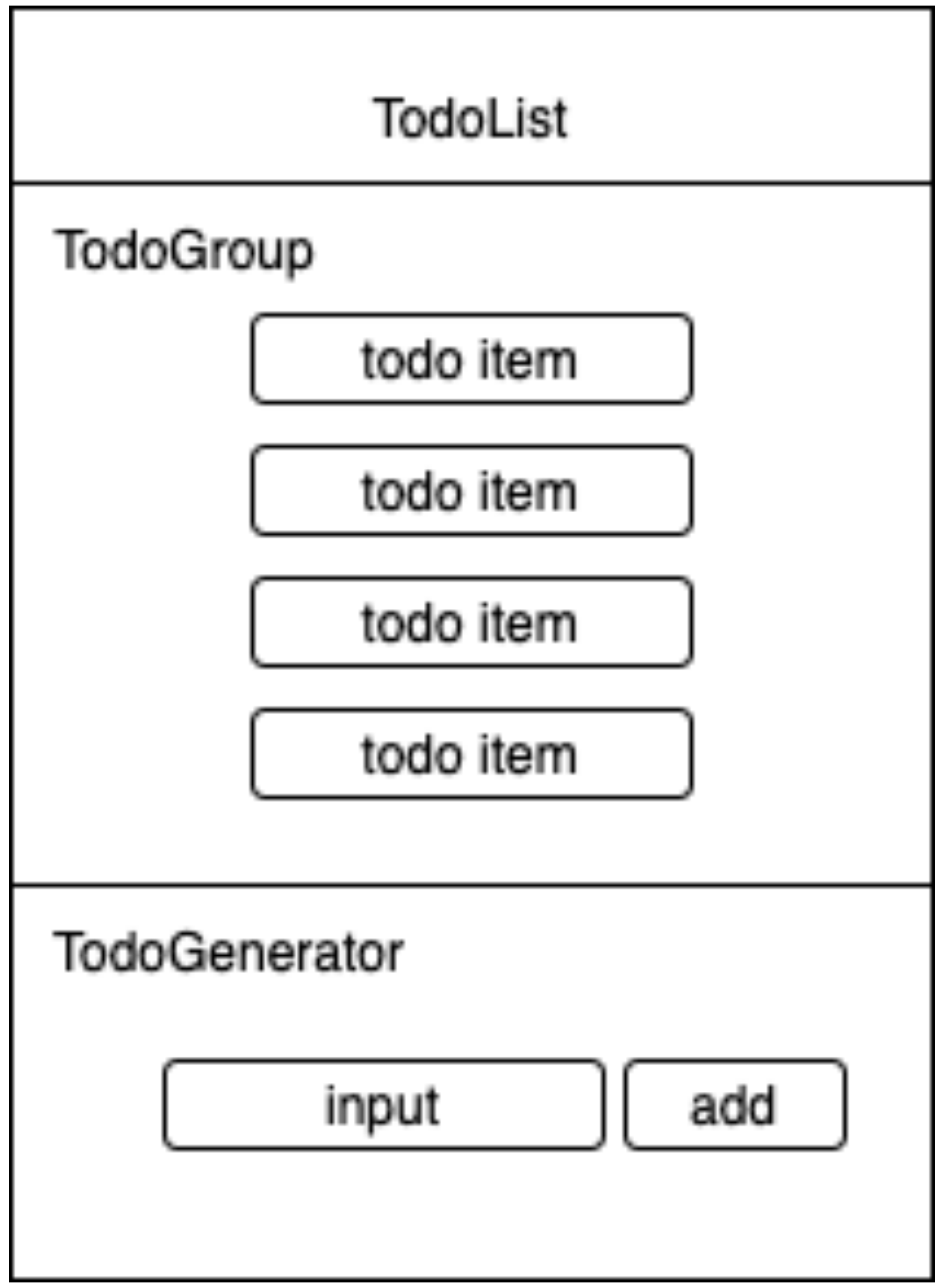
3. Toggle a todo item

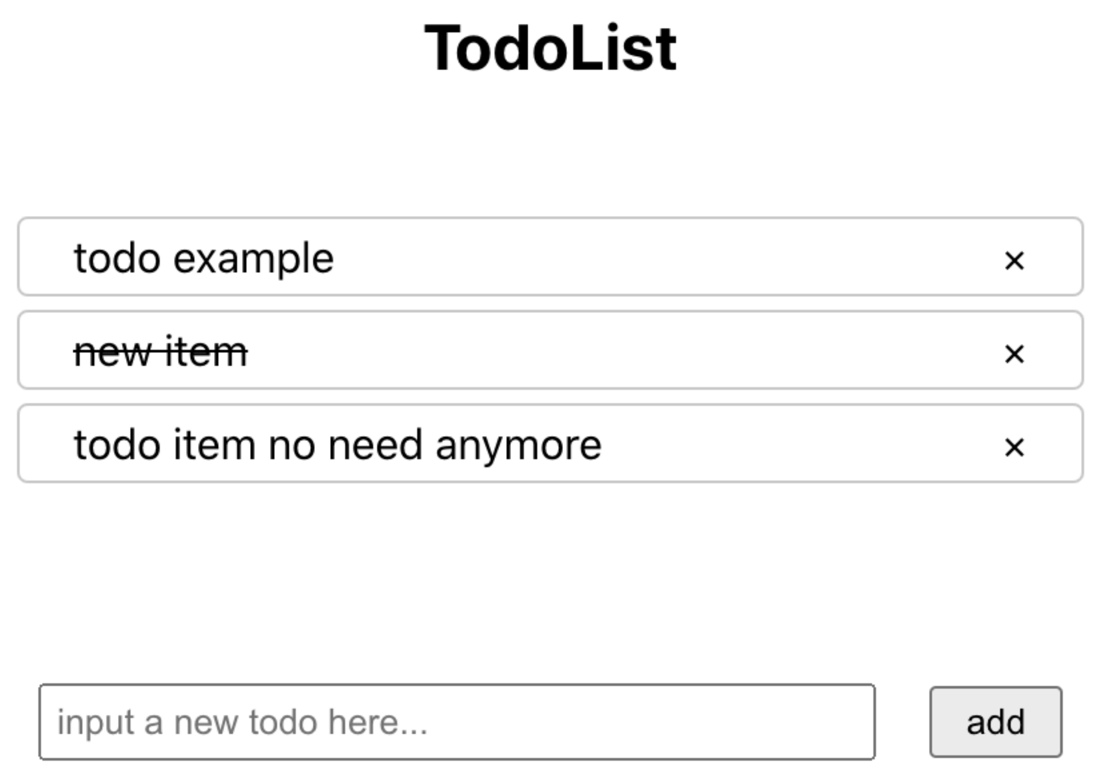
4. **Delete a todoitem**
   - Show a “X” icon behind todo item
   - Add event handler when click “X” icon
   - Update the global store to update the clicked item status





TodoList Components





TodoList

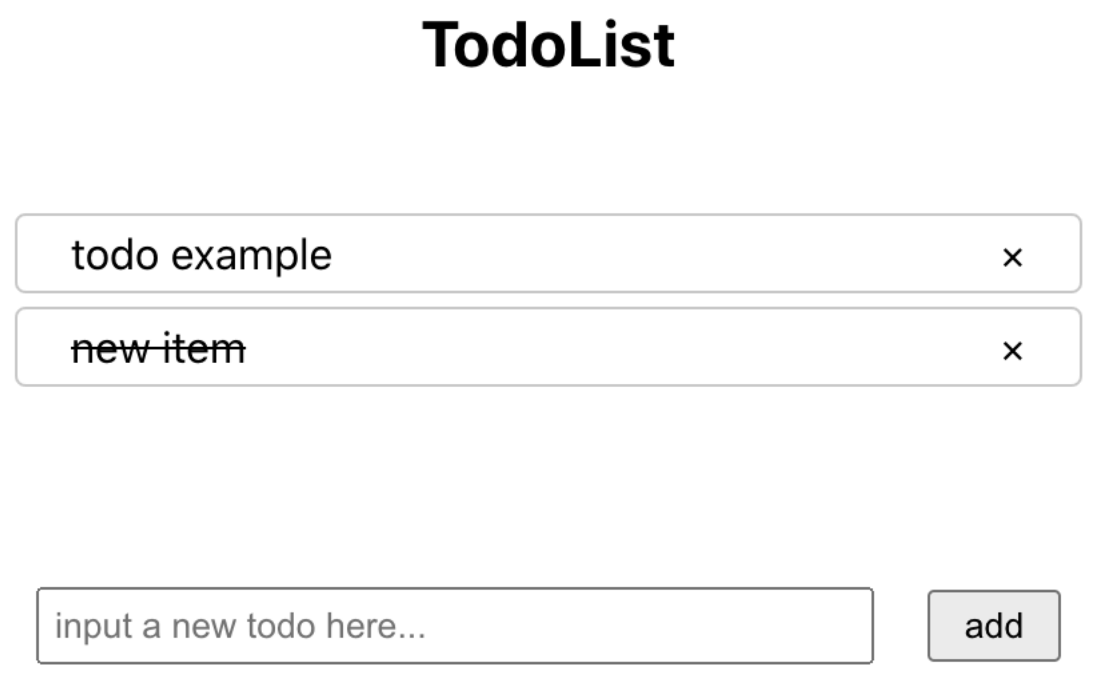
todo example ×

~~new item~~ ×

todo item no need anymore ×

input a new todo here... | add

```
[
  {
    id: "cc53dc26-61b0-406b-99dd-b8825dd2ceec"
    text: "todo example"
    done: false
  },
  {
    id: "dd53dc26-b061-6b40-dd99-82b85dd2ce90"
    text: “new item"
    done: true
  },
  {
    id: "df53dc26-b061-6b40-dd99-82b85dd2ce98"
    text: “todo item no need anymore"
    done: false
  }
]
```



TodoList

todo example ×

~~new item~~ ×

input a new todo here... | add

```
[
  {
    id: "cc53dc26-61b0-406b-99dd-b8825dd2ceec"
    text: "todo example"
    done: false
  },
  {
    id: "dd53dc26-b061-6b40-dd99-82b85dd2ce90"
    text: “new item"
    done: true
  }]
```

Homework

Deadline 22:00pm

1. **Complete the todo list**

- Toggle a todo item

- Add event handler when click one todo item
- Update the global store to update the clicked item status
- Add style for the completed todo item

- Delete a todoitem

- Show a “X” icon behind todo item
- Add event handler when click “X” icon
- Update the global store to update the clicked item status

2. Learn about **Promise** Syntax, and implement a code demo with how to use Promise

3. Learn about **async** and sync JavaScript and know the difference

4. **Diary** with ORID