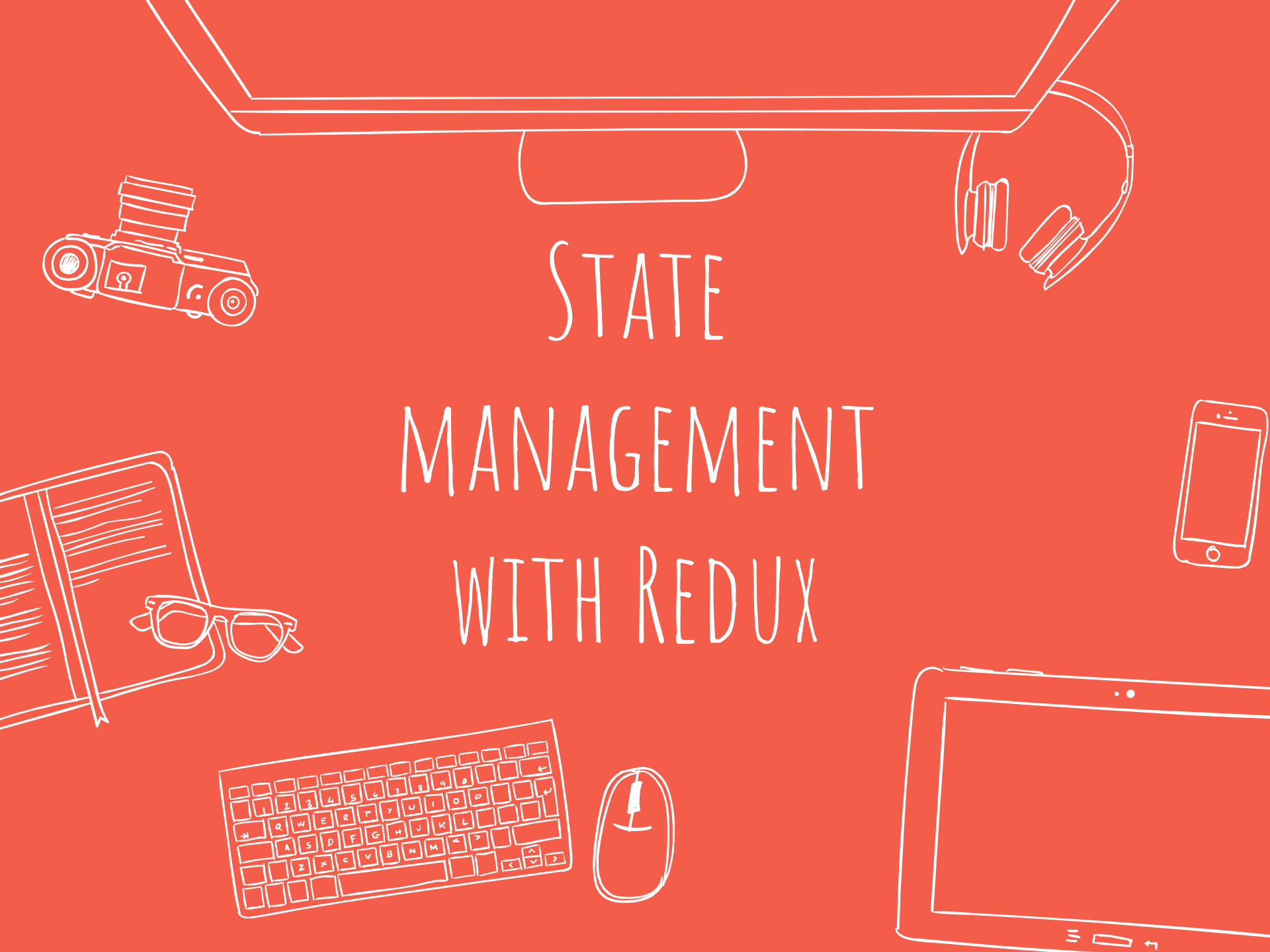
State Management with Redux



1.

Why Redux?

Challenges with the state

Components are the building blocks of a React UI:

- ✖ Each component can maintain **its own state**
- ✖ State needs to be communicated to child components through **props**

Challenges with the state

Usually the state needs to be maintained in the **top level parent component.**

What happens when there are many levels deep of child components?

The information flow becomes a problem as the UI is getting more complex.

Challenges with the state

In larger applications, a lot of data is moving through unrelated component:

- ✖ passed down via props or
- ✖ passed up using callbacks

What the solution?

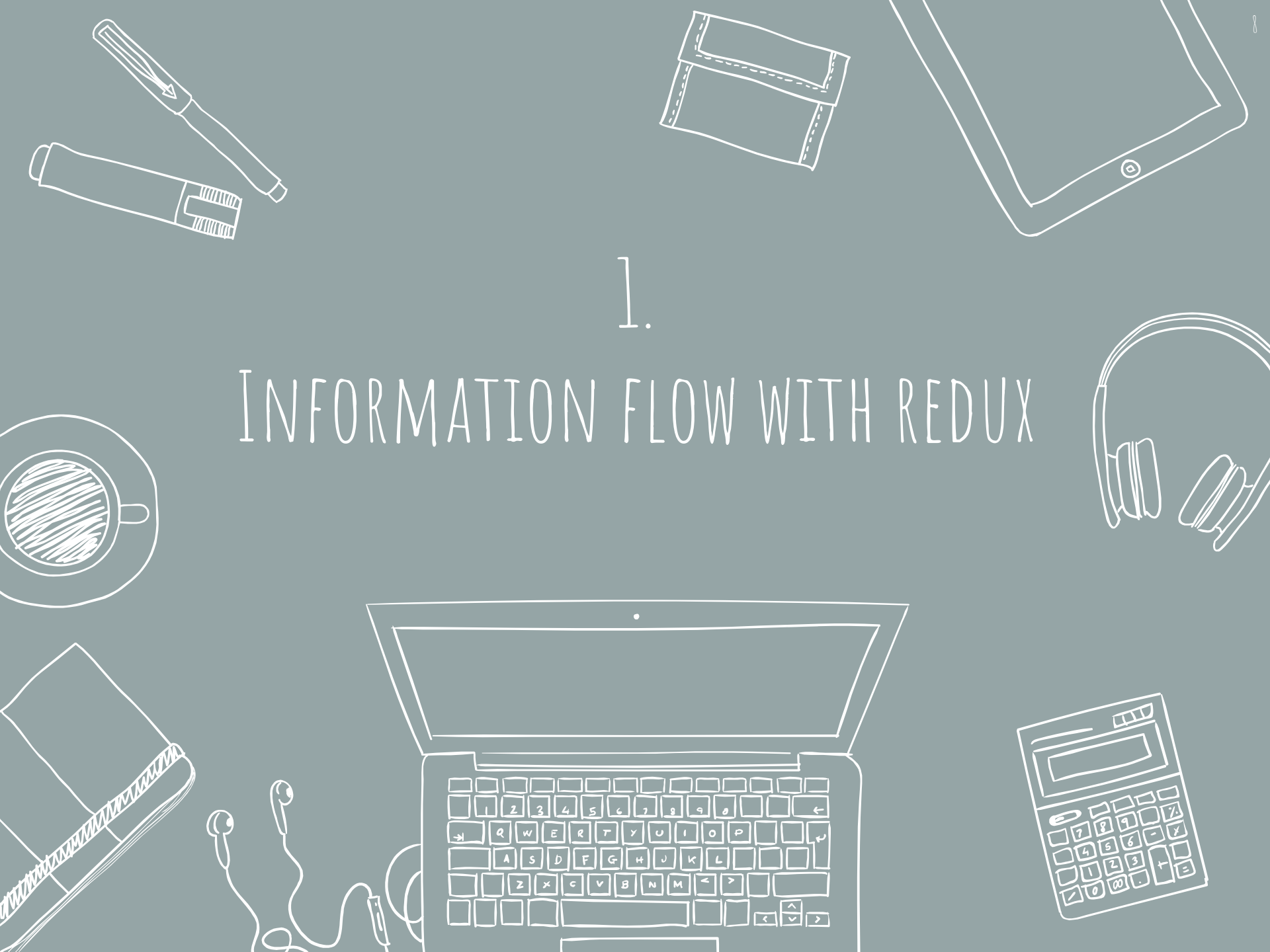
One solution is to manage the state in a single location with Redux.

Why Redux?

One solution is to manage the state in a single location with Redux.

- Redux stores the entire application state in a single object known as *the source of truth.*

- *Redux is a state management library that goes well with React, but it can be used by any application.*



1.

Information flow with redux

Information Flow with Redux

With Redux, you never modify the state directly.

To manage the state, Redux introduces the following concepts:

- ✖ Store
- ✖ Actions
- ✖ Reducers

Information Flow with Redux

Store

- The container that maintains the state of the app

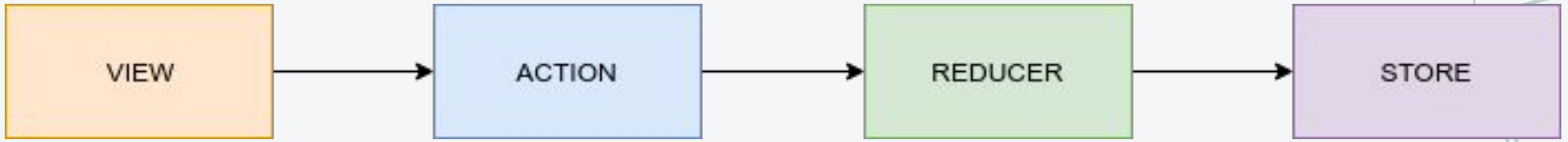
Actions

- To change the state, you need to dispatch actions that describes what needs to happen.

Reducers

- Functions that are responsible for updating the state.

Information Flow with Redux



An event happens

Describe what needs to happen

How to update the state

Maintains the state

The Redux Store

To create a Redux Store:

You create a Redux store with the createStore function:

createStore(reducer, initialState, middleware)

Middleware such as *logger* allows us to console.logs the changes in the state.

The Redux Store

To give all the components access to the Redux store, we're going to use Provider (React Context API) + connect from react-redux:

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

Any component can have access to store through props.

The Redux Store

The Redux store gives us access to the following method:

- store.getState() - Returns the current state.
- store.dispatch(action) - Dispatch an action to change the state.
- store.subscribe(listener) - Listen to changes in the state tree.

ACTIONS

Actions are functions that returns an object with the type of action and the property to update the state:

```
const receiveTweetsAction = {
  type: RECEIVE_TWEETS,
  tweets
}
```

ACTIONS CREATORS

Typically the object that defines an action is returned by a function:

```
const RECEIVE_TWEETS = "RECEIVE_TWEETS";

function receiveTweets(tweets) {
 return {
   type: RECEIVE_TWEETS,
   tweets
 };
}
```

Async Calls

Async calls such as retrieving the tweets from the database were being performed in component did mount.

With Redux, the code logic is moved into an action creator that returns a function.

Async Calls: Getting the Tweets

```
function handleInitialData() {
 return dispatch => {
  return fetch("/tweets")
    .then(res => res.json())
    .then(tweets => dispatch(receiveTweets(tweets)));
 };
}
```

Reducers

Reducers are responsible for updating part of the state, depending on the type of action:

```
const RECEIVE_TWEETS = "RECEIVE_TWEETS";

function tweets(state = {}, action) {
 switch (action.type) {
   case RECEIVE_TWEETS:
     return {
       ...state,
       ...action.tweets
     };
   default:
     return state;
 }
```

Thanks!

Any questions?

You can find me at:

- twitter.com/DCTremblay
- in/dominictremblay/