

[https://codesandbox.io/s/lecture-17-9d6jzz?  
file%253D%252Fsrc%252FApp.js](https://codesandbox.io/s/lecture-17-9d6jzz?file%253D%252Fsrc%252FApp.js)

# Async in Redux

- What are three Redux principles?
- If the update of state need to involve async operations, how to do it?
- Why difficult?
  - Async and Promise -> Unpredictable

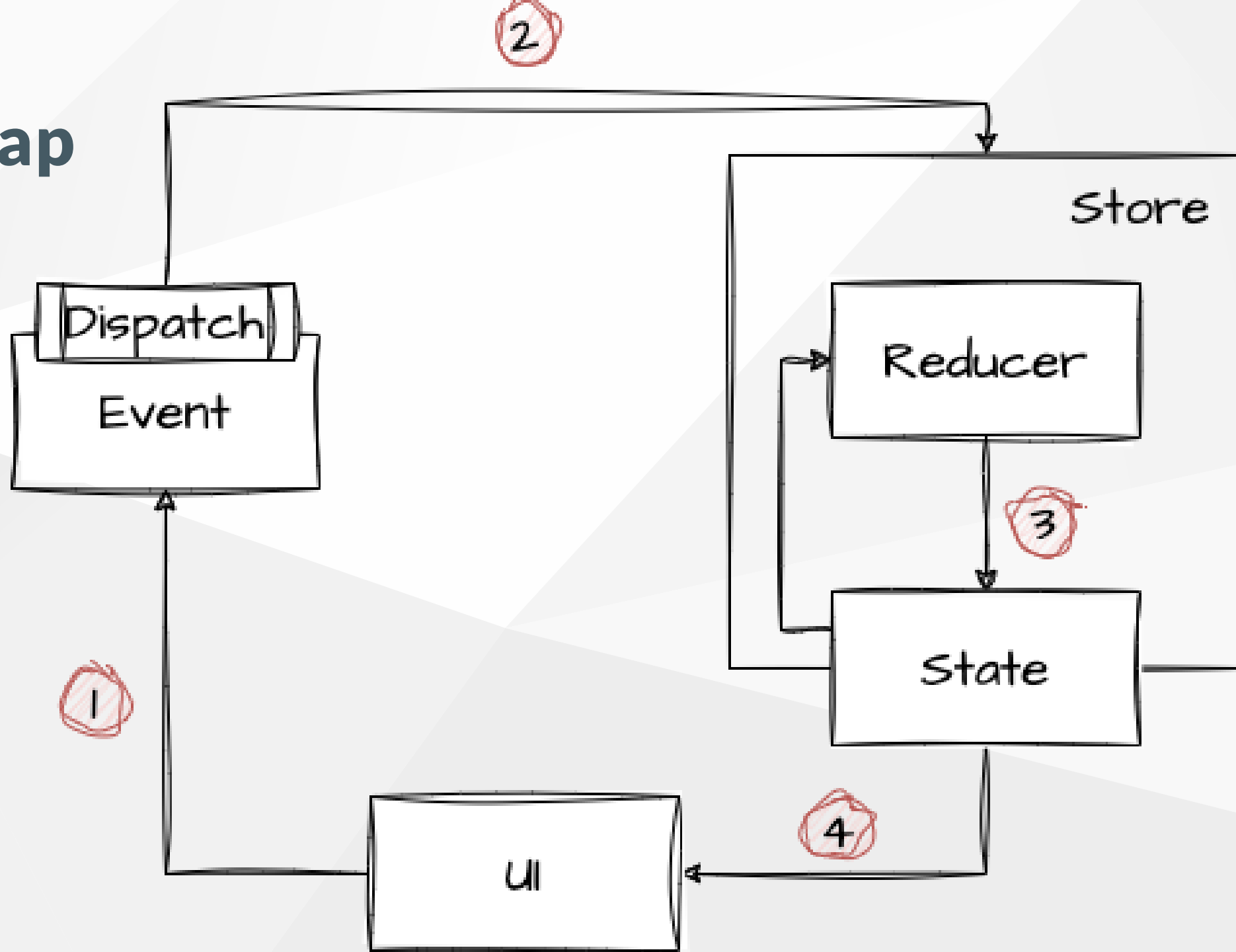
# REST API with React

- `fetch` API / `axios`
- `useEffect` hook / event handler
- lifecycle methods, i.e. `componentDidMount`

# REST API with Redux

- Redux Thunk
- Redux Saga

# Recap



# Action Creator

```
// sync action creator  
const fetchUser = () => {  
  const users = getUsersSync();  
  return {  
    type: 'FETCH_USER',  
    payload: users  
  };  
};
```

```
// async action creator  
const fetchUser = async () => {  
  const users = await getUsersAsync();  
  return {  
    type: 'FETCH_USER',  
    payload: users  
  };  
};  
// Not possible with native Redux
```

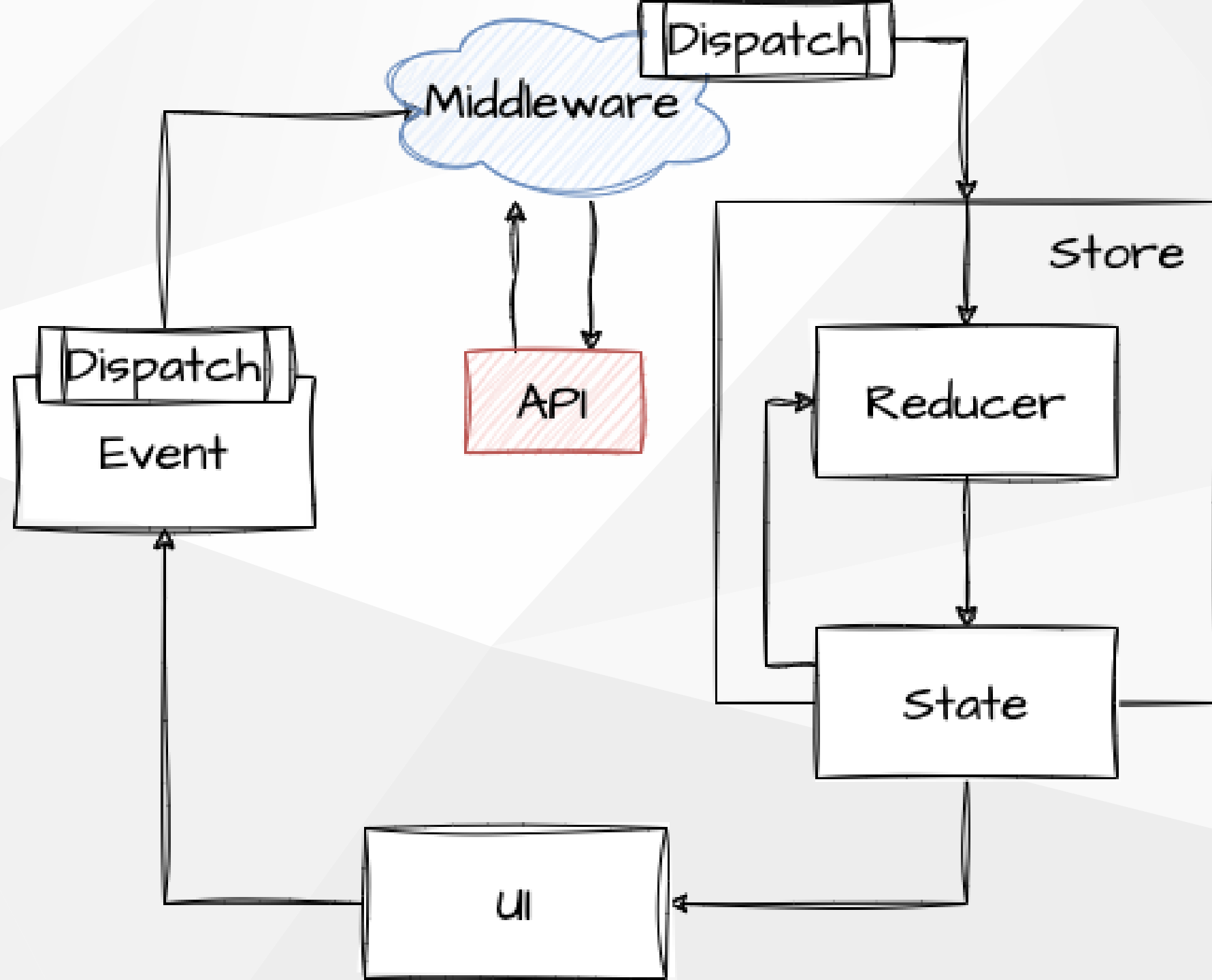
# Redux Thunk

- Redux Thunk is a middleware that allows you to return functions, not just actions, within Redux
- `redux-thunk`

# Redux Middleware

- Redux middleware is a function that is able to intercept, and act accordingly, our actions, before they reach the reducer
- Executed between dispatching an action and the moment it reaches the reducer
- Multiple middleware can be chained together





# Middleware Example

```
const logger = store => next => action => {  
  console.log('dispatching', action);  
  let result = next(action);  
  console.log('next state', store.getState());  
  return result;  
};
```

# Thunk

```
function createThunkMiddleware(extraArgument) {  
  const middleware =  
    ({ dispatch, getState }) =>  
      next =>  
        action => {  
          // The thunk middleware looks for any functions that were passed to `store.dispatch`.  
          // If this "action" is really a function, call it and return the result.  
          if (typeof action === 'function') {  
            // Inject the store's `dispatch` and `getState` methods, as well as any "extra arg"  
            return action(dispatch, getState, extraArgument);  
          }  
  
          // Otherwise, pass the action down the middleware chain as usual  
          return next(action);  
        };  
  return middleware;  
}
```

# Refactor Action Creator

```
const fetchUser = () => {  
  const users = getUsersSync();  
  return {  
    type: 'FETCH_USER',  
    payload: users  
  };  
};
```

```
const fetchUser = () => {  
  return async (dispatch, getState) => {  
    const users = await getUsersAsync();  
    dispatch({  
      type: 'FETCH_USER',  
      payload: users  
    });  
  };  
};
```

# Handle Promise Status

- `pending`
- `fulfilled`
- `rejected`
- how to handle them with UI and Redux?

# Redux Thunk with RTK

features

# RTK Query vs React Query

- RTK Query and React Query is a data fetching and caching tool
- RTK Query is built on top of Redux, while React Query is built on top of React Hooks
- RTK Query is more opinionated, while React Query is more flexible

# RTK Query

- `createApi`
- `useGetQuery`
- `useMutation`



# Additional Thoughts

- React state still needed?
  - Yes, for local state, i.e. form input
  - Redux is for global state management
- What data should be put in Redux?
  - Data that is shared across components
  - Any piece of data that changes over time
- How to structure Redux store / reducers and their dispatch?
  - [Redux Style Guide](#)

# Redux Recap

1. Create a `store` (single source of truth)
2. Link `store` to root component ( `Provider` )
3. Create `reducer`s (pure functions)
4. Create `action creator`s (functions that return actions)
5. Connect `action creator`s to `dispatch` (dispatch actions)

Note: In class component, we need to use `connect` to connect `store` to specific components. In functional component, we use `useSelector` and `useDispatch` hooks, instead.

# Testing in React

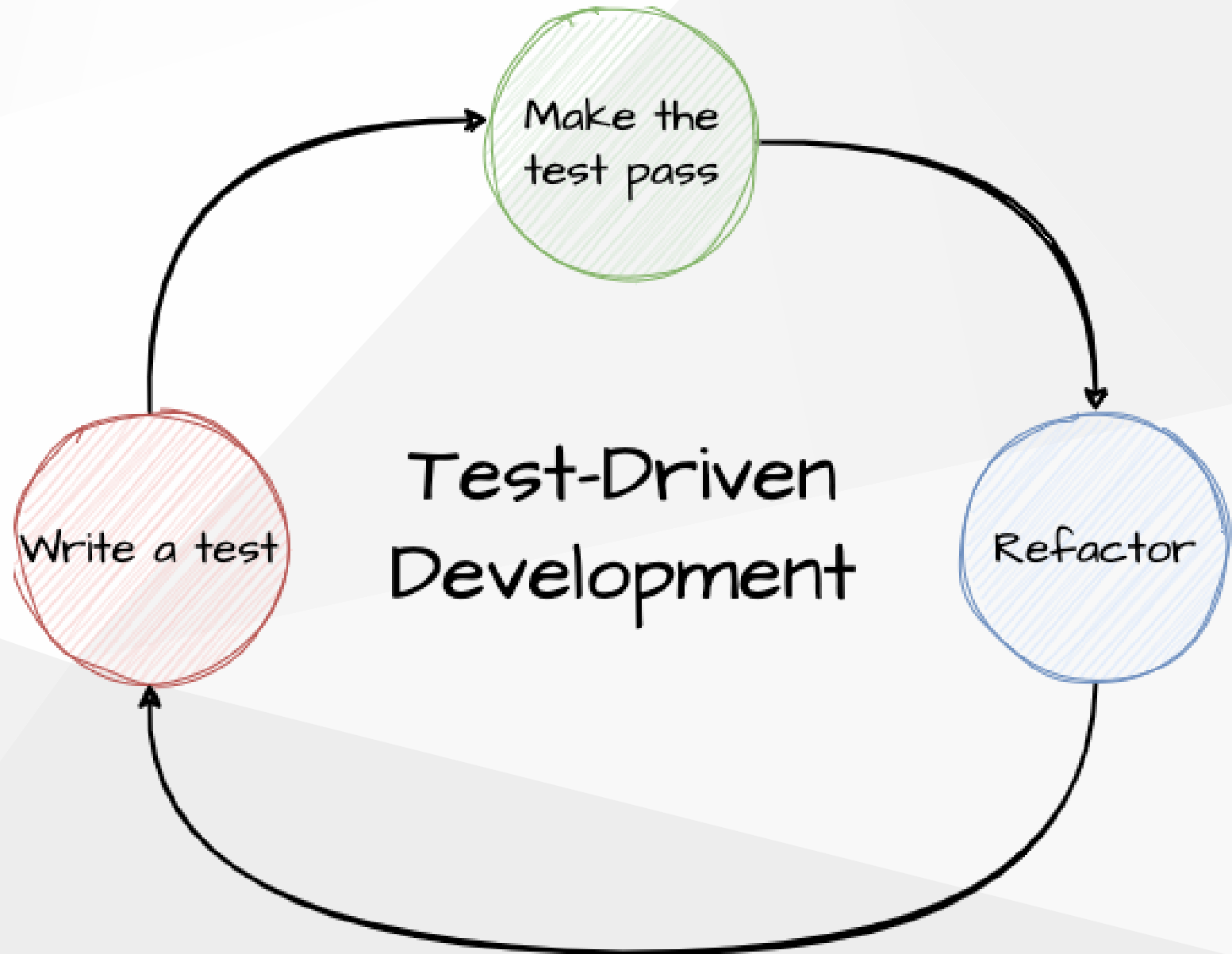
- What is testing?
- Why testing?
- How to test?

# What is testing?

- Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not
- Unit testing
- Integration testing
- End-to-end testing

# Test-Driven Development (TDD)

1. Write tests first
2. Write code to pass the tests
3. Refactor



# Why testing?

- Reduce bugs in existing/new features
- Improve code quality
- Reduce cost of change
- Improve design
- Improve team confidence

# How to test

- Unit testing
  - [React Testing Library](#)
  - [Enzyme](#)
  - [Jest](#)
- E2E testing
  - [Cypress](#)
  - [Playwright](#)