“Classic” approach
State-based

```
class DeleteConfirm extends React.Component {
  state = {
    showConfirm: false
  };
  render() {
    return this.state.showConfirm ? (
      <React.Fragment>
        <button
          onClick={() => {
            this.setState({
              showConfirm: false
            });
          }}
        >
          Confirm delete?
        </button>
        <button
          onClick={() => {
            this.setState({
              showConfirm: false
            });
          }}
        >
          Cancel
        </button>
      </React.Fragment>
    ) : null;
  }
}
```

Scenario-based

```
async function* DeleteConfirm() {
  while (true) {
    await waitFor(types.DELETE_CLICKED)

    yield (
      <React.Fragment>
        <button
          onClick={() => {
            store.dispatch({
              type: types.CONFIRM_DELETE
            })
          }}
        >
          Confirm delete?
        </button>
        <button
          onClick={() =>
            store.dispatch({
              type: types.CANCEL_DELETE
            })
          }
        >
          Cancel
        </button>
      </React.Fragment>
    )

    await waitFor(types.CONFIRM_DELETE, types.CANCEL_DELETE)
    yield null
  }
}
```

Do we even need Redux's state or React's state when we can use generators state?

```
async function* Counter() {
  let counter = 0;
  while (true) {
    yield (
      <section>
        <p>{counter}</p>
        <button onClick={() => dispatch('INCREMENT')}>Increment</button>
      </section>
    );
    await waitFor('INCREMENT');
    counter++;
  }
}
```