RxJS with React

The best, most maintainable code, is code that matches the way you think about the problem, without a lot of noise

What do I mean by noise…?

Therefore, *the best*

Language / Framework / Library

FOR YOU

is the one that either:

A. Matches your way of thinking

B. Guides you to think in a different way

Succinct, expressive code, that lets you build complex web applications

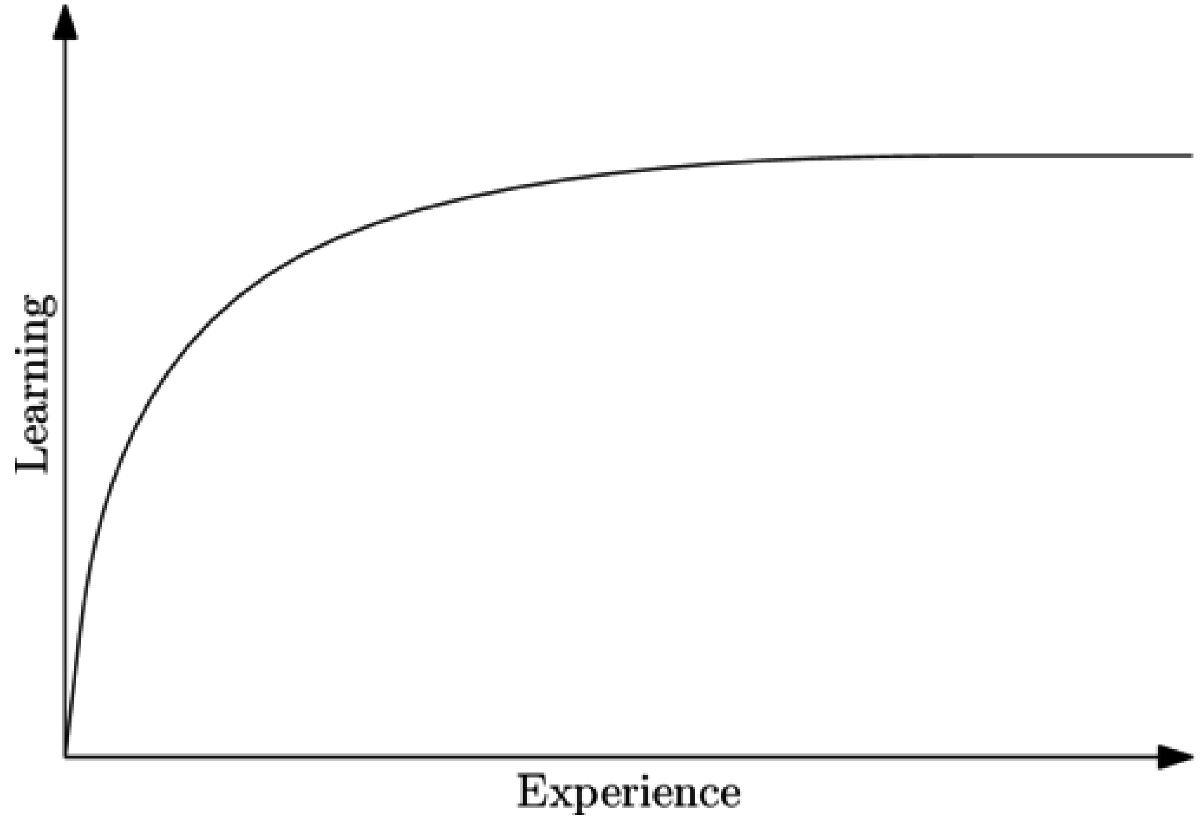
"without a lot of noise"

Why RxJS?

Succinct, expressive code for dealing with

events, asynchronicity and state

Steep Learning Curve



Declarative Style

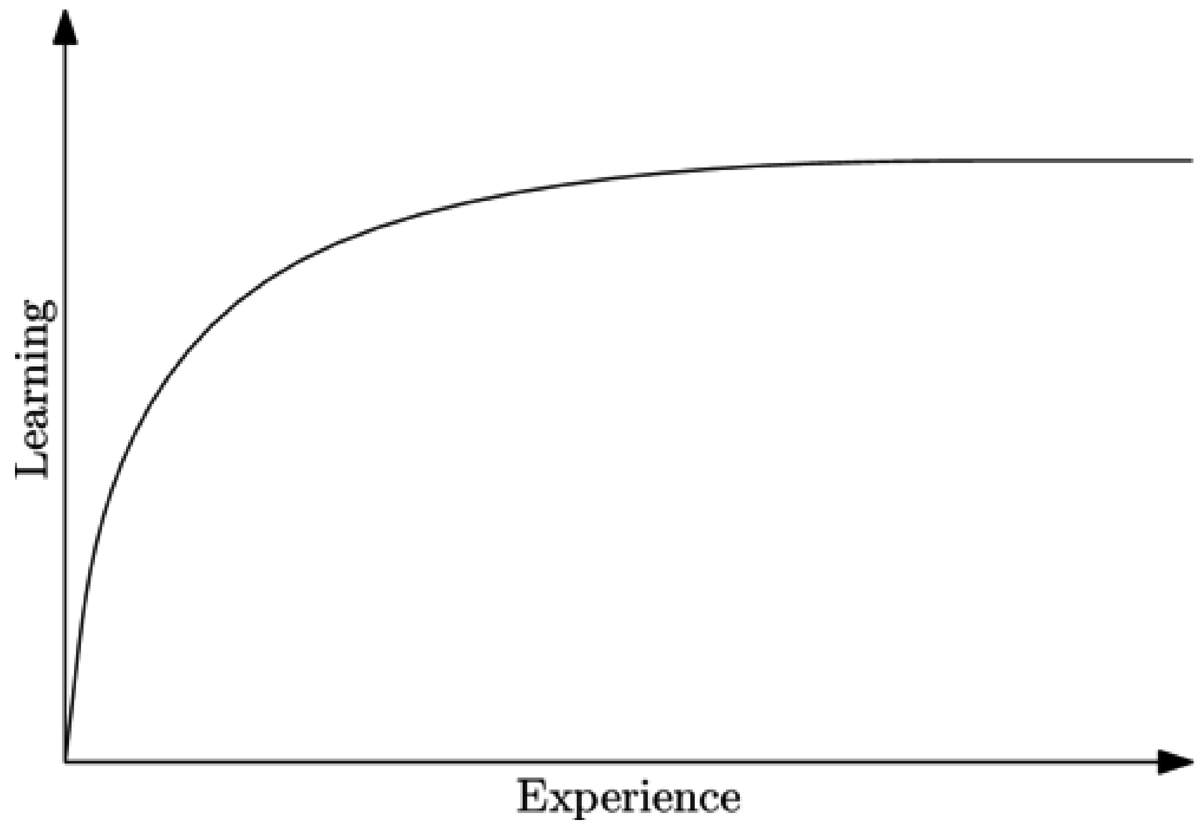
Functional in Nature

Encourages Immutability

Why React?

Succinct, expressive code for dealing with **UI based on discrete synchronous state updates**

Steep Learning Curve

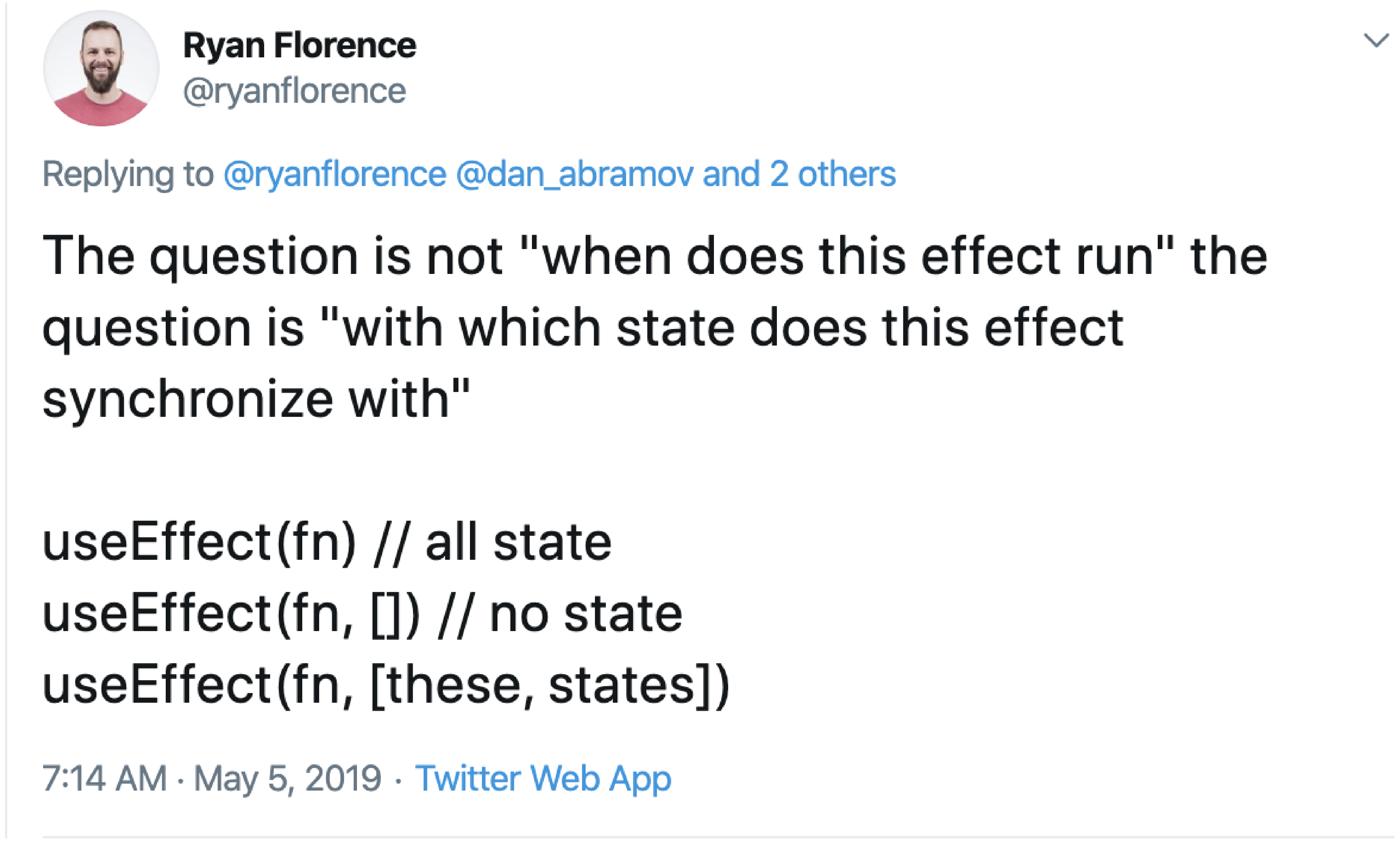


Declarative Style

Functional in Nature

Encourages Immutability

Why RxJS with React?



Ryan Florence
@ryanflorence

Replying to @ryanflorence @dan_abramov and 2 others

The question is not "when does this effect run" the question is "with which state does this effect synchronize with"

```
useEffect(fn) // all state
useEffect(fn, []) // no state
useEffect(fn, [these, states])
```

7:14 AM · May 5, 2019 · Twitter Web App

Butter

Change the way you think about problems

How to use RxJS with React

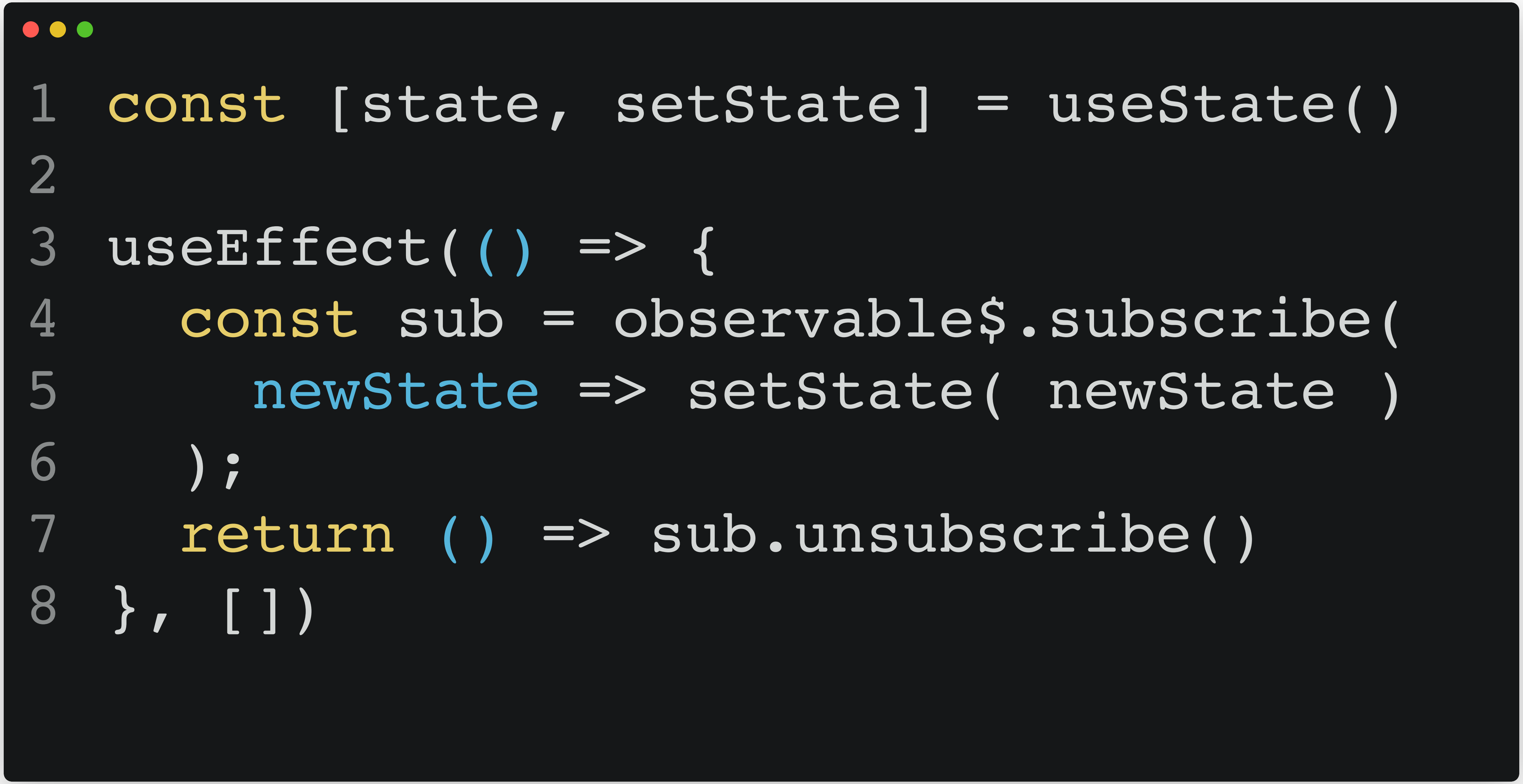
observables with

`useState()`

+

`useEffect()`

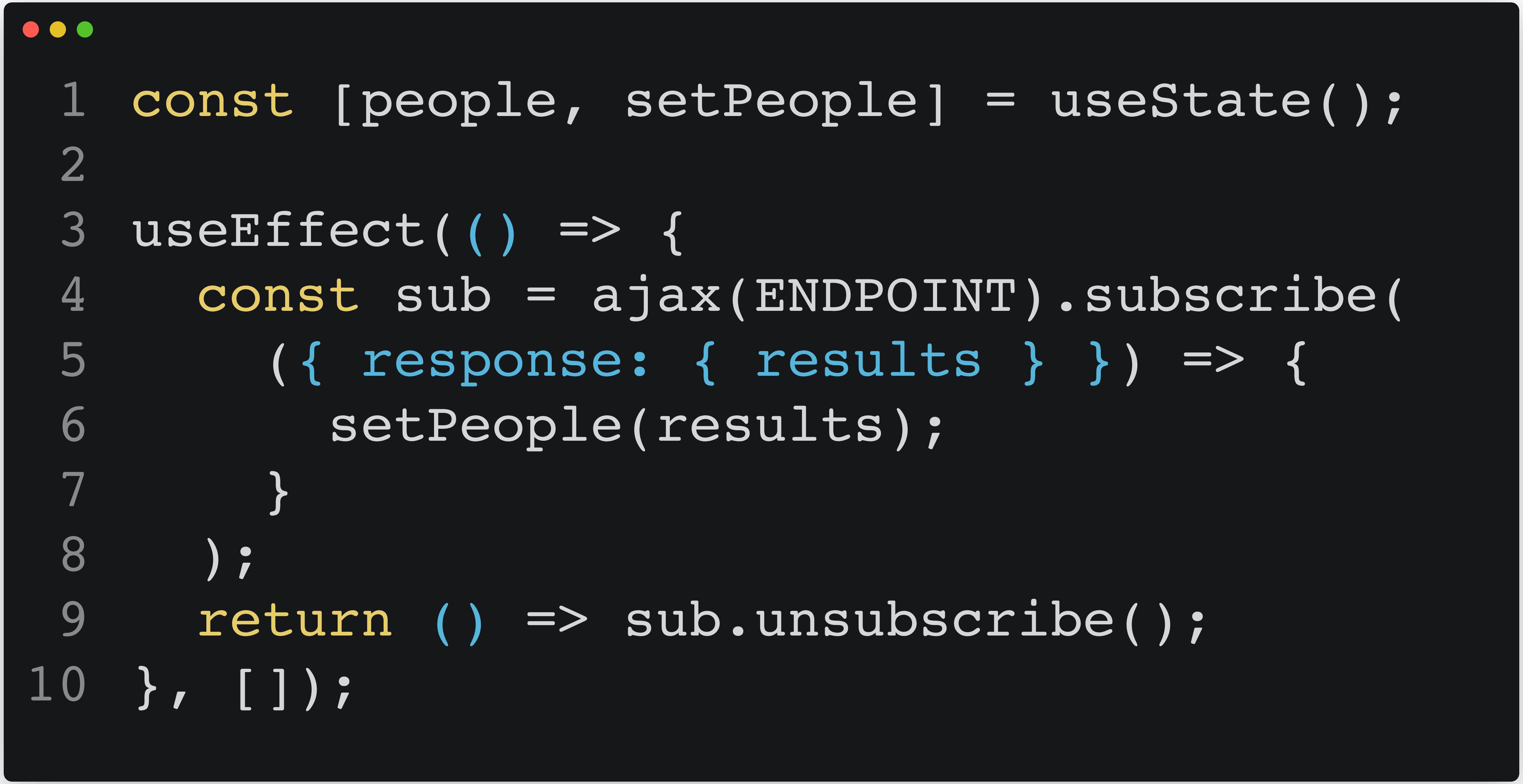
Syncing State



```
const [state, setState] = useState()

useEffect(() => {
  const sub = observable$.subscribe(
    newState => setState( newState )
  );
  return () => sub.unsubscribe()
}, [])
```

RxJS.ajax()



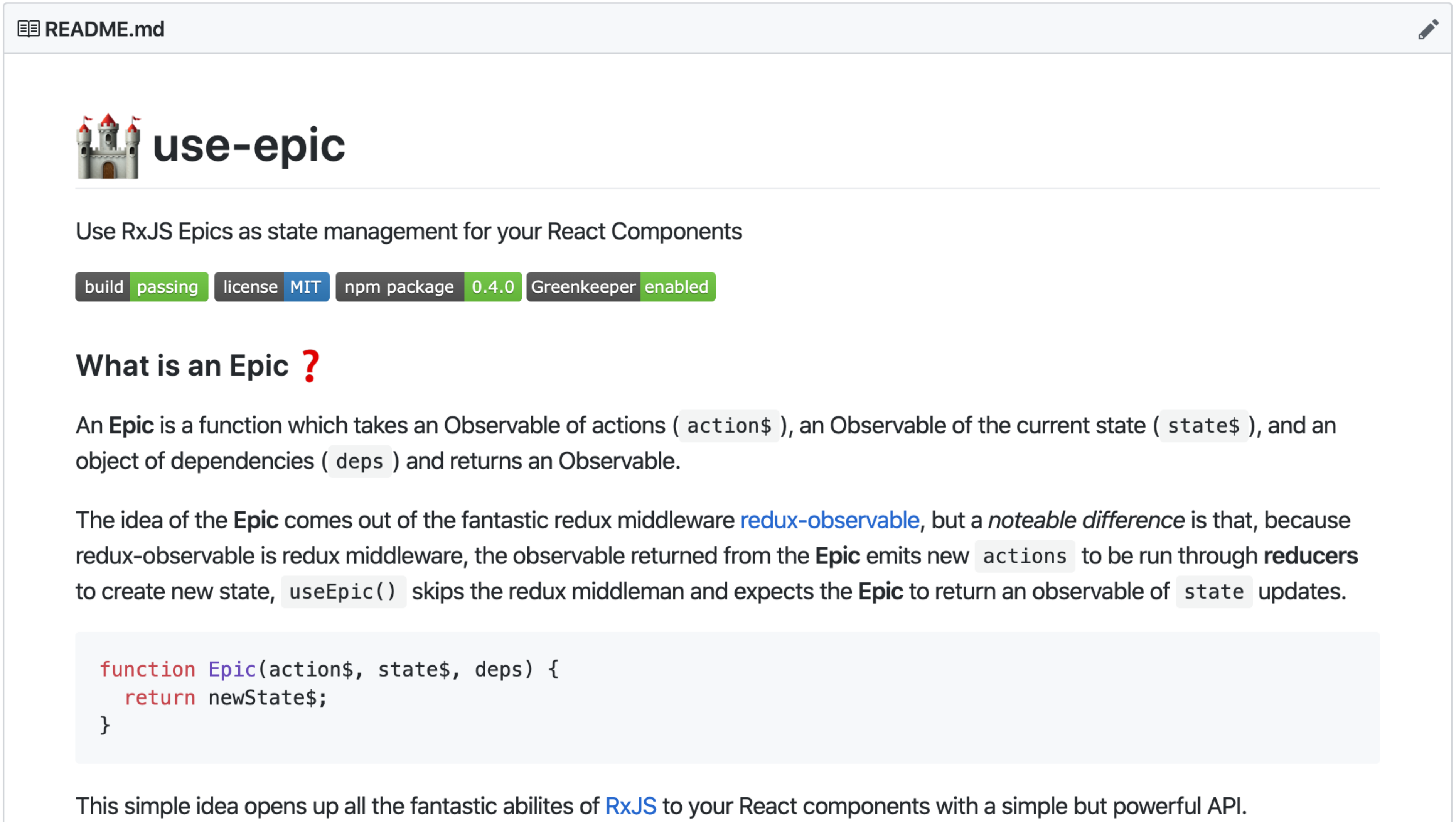
```
const [people, setPeople] = useState();

useEffect(() => {
  const sub = ajax(ENDPOINT).subscribe(
    ({ response: { results } }) => {
      setPeople(results);
    }
  );
  return () => sub.unsubscribe();
}, []);
```

+

useEpic

https://github.com/BigAB/use-epic



README.md

🏰 use-epic

Use RxJS Epics as state management for your React Components

build passing | license MIT | npm package 0.4.0 | Greenkeeper enabled

What is an Epic ❓

An **Epic** is a function which takes an Observable of actions (`action$`), an Observable of the current state (`state$`), and an object of dependencies (`deps`) and returns an Observable.

The idea of the **Epic** comes out of the fantastic redux middleware redux-observable, but a *noteable difference* is that, because redux-observable is redux middleware, the observable returned from the **Epic** emits new `actions` to be run through **reducers** to create new state, `useEpic()` skips the redux middleman and expects the **Epic** to return an observable of `state` updates.

```
function Epic(action$, state$, deps) {
  return newState$;
}
```

This simple idea opens up all the fantastic abilites of RxJS to your React components with a simple but powerful API.

Benefits?

Succinct, expressive code

Powerful best in class abstractions

Separation of Concerns

Components become about User Interface

Separation of Concerns

Observables are how you handle application state and events

Testing

Test components and observables separately

Cross Framework

Use tools and from other communities

Awesome Rx Tools

RxJS is super popular across frameworks so, sometimes we get to use solutions from other communities, and there is just a lot of cool RxJS stuff out there

- <$> React RxJS Elements
  https://github.com/kosich/react-rxjs-elements
- React-RxJS
  https://react-rxjs.org/
- RxJS State
  https://github.com/BioPhoton/rxjs-state
- Rx-Handler
  https://github.com/johnlindquist/rx-handler
- Rx-Query
  https://github.com/timdeschryver/rx-query
- Sanity.io
  https://www.sanity.io/docs/client-libraries/js-client
- Rx Visualizer
  https://rxviz.com

RxJS with React

BigAB

@adamlbarrett

Anedot