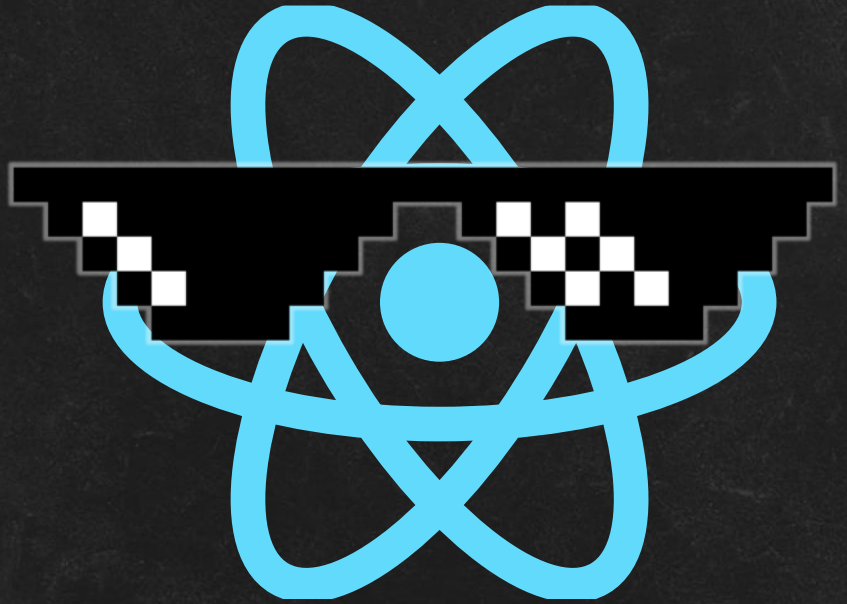


Why do you need to...

- Use unidirectional data flow
- Avoid calling hooks conditionally
- Store state using the `useState` hook

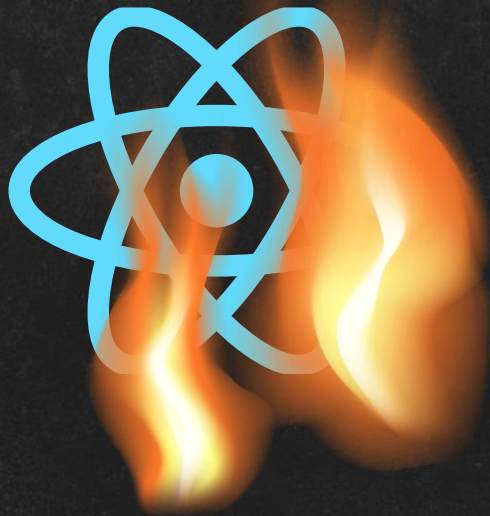
- Conditionally call React hooks
- Build impure components
- Pull in a second UI library

About me

- Senior Software Engineer building Microsoft Loop
- Worked on Edge, Chrome, and Firefox
- React developer for 8+ years

tigeroakes.com
@notwoods.bsky.social
@not_woods

The art of ignoring best practices for React performance



react@18.2.0

Overview

Hooks

Components

APIs

Directives

react-dom@18.2.0

Hooks

Components

APIs

Client APIs

Rules of Hooks

Hooks are defined using JavaScript functions, but they represent a special type of reusable UI logic with restrictions on where they can be called. You need to follow the Rules of Hooks when using them.

- Only call Hooks at the top level – Don't call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function, before any early returns.
- Only call Hooks from React functions – Don't call Hooks from regular JavaScript functions.

NEXT
Components and Hooks must be pure

CAUTION!

Use your best judgement, evaluate tradeoffs, and communicate.

Photo by Markus Spiske on Unsplash

THE PLAN

01

When does React rerender?

02

How to isolate expensive hooks

Using impure components

03

How to only update leaf components

Using a different UI library

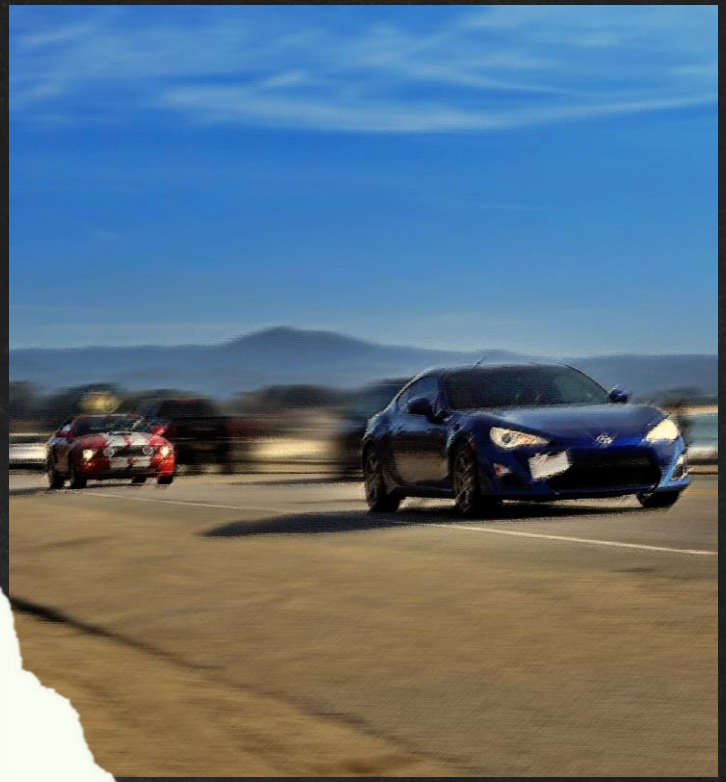
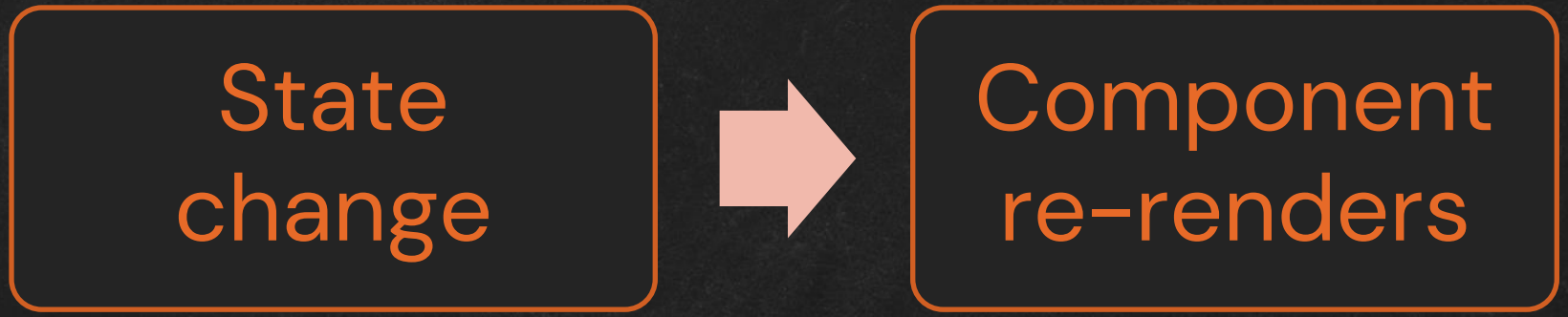


Photo by Noah Boyer on Unsplash

01

When does React rerender?



Every rerender starts with a state change.

Every rerender starts with a state change.

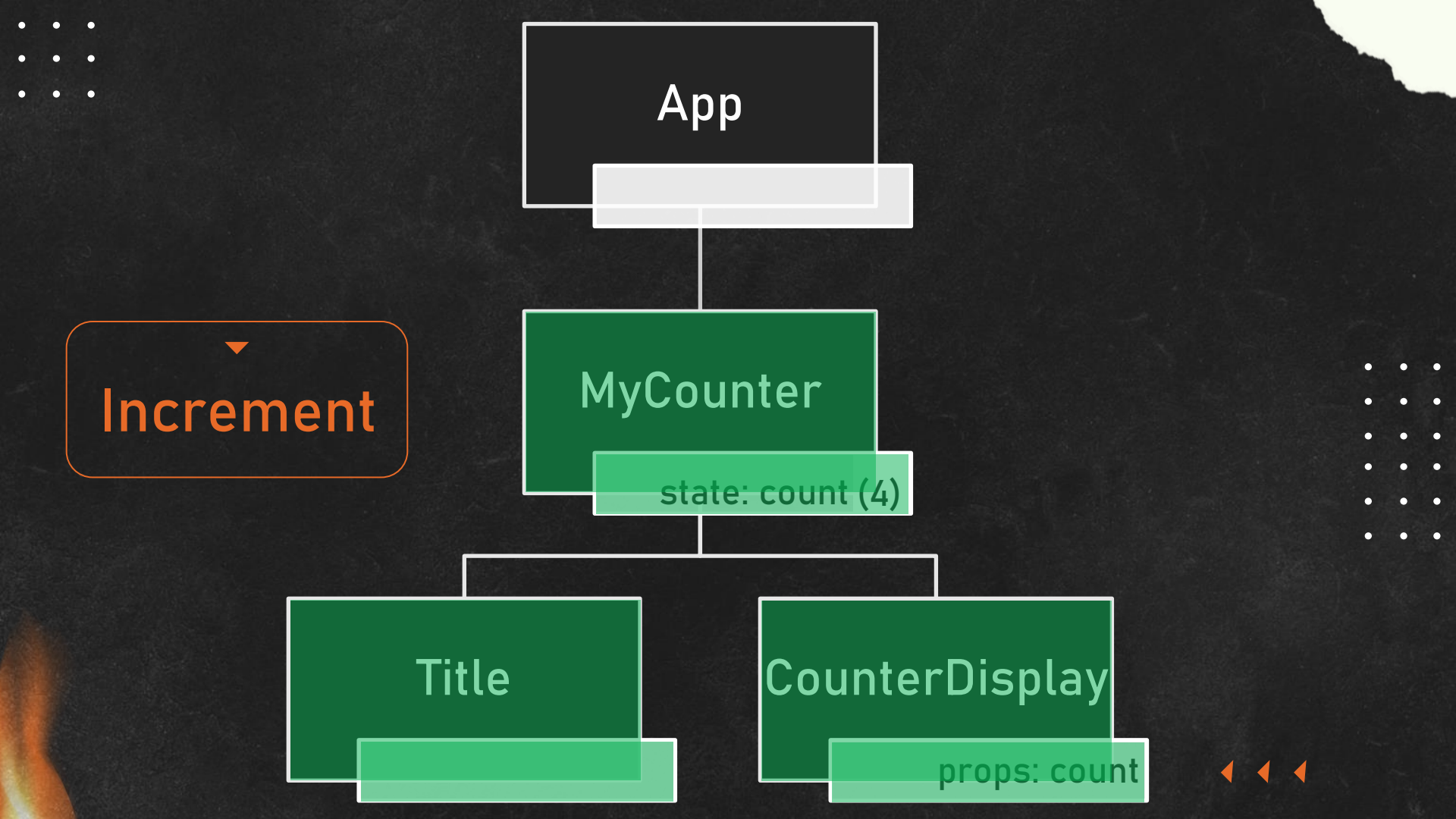
```jsx
function MyCounter() {
  const [count, setCount] = useState(0);
  const handleClick = () => setCount(count + 1);
  return (
    <div>
      <Title />
      <CounterDisplay count={count} />
      <button onClick={handleClick}>Increment</button>
    </div>
  );
}

function Title() {
  return <h1># of rules broken</h1>;
}

function CounterDisplay({ count }) {
  return <strong>{count}</strong>;
}
```

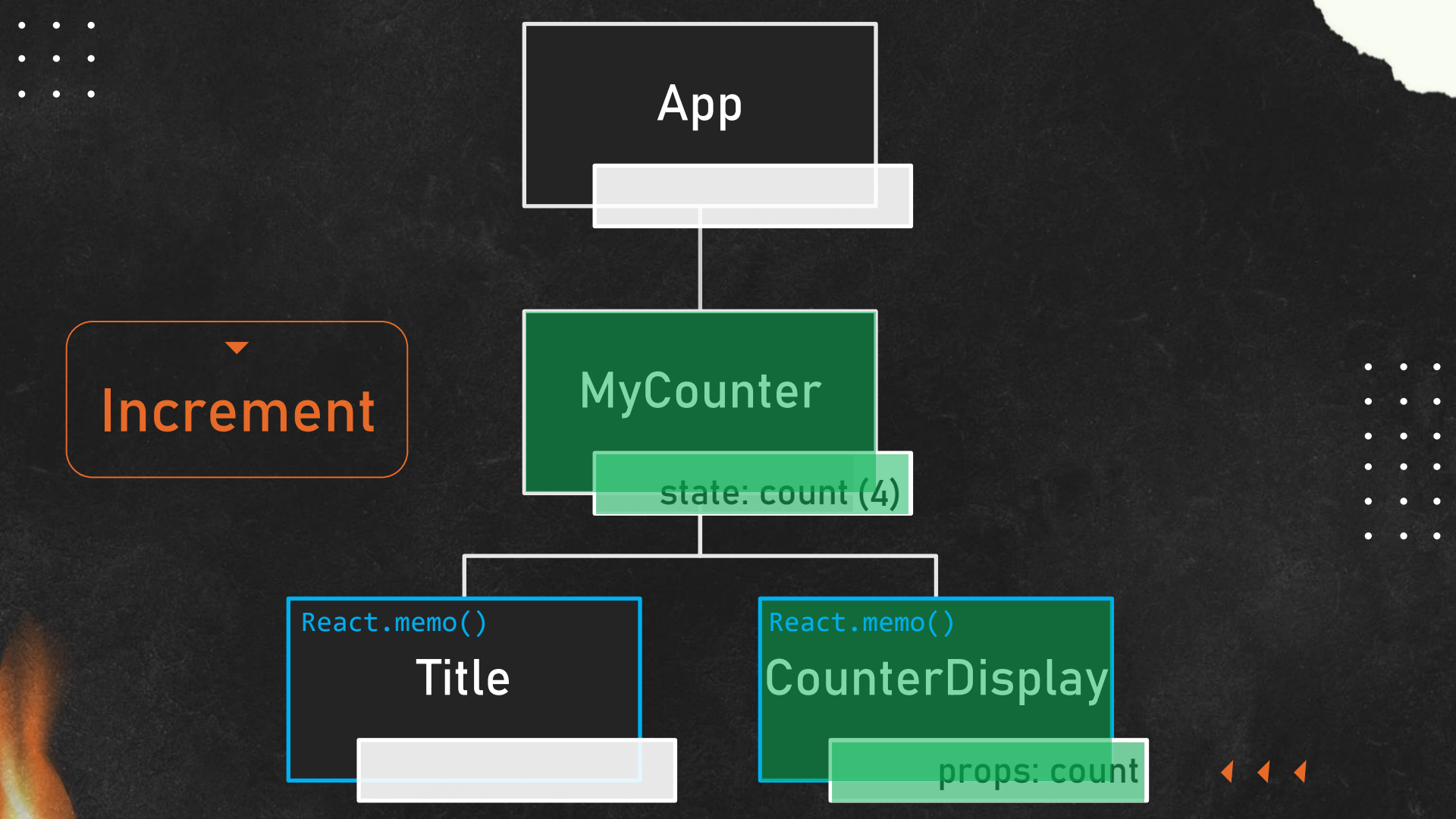
```
function MyCounter() {
  const [count, setCount] = useState(0);
  const handleClick = () => setCount(count + 1);
  return (
    <div>
      <Title />
      <CounterDisplay count={count} />
      <button onClick={handleClick}>Increment</button>
    </div>
  );
}

function Title() {
  return <h1># of rules broken</h1>;
}

function CounterDisplay({ count }) {
  return <strong>{count}</strong>;
}
```

```
function MyCounter() {



    <div>
      <h1># of rules broken</h1>
      <strong>{1}</strong>
      <button onClick={handleClick}>Increment</button>
    </div>

}
```

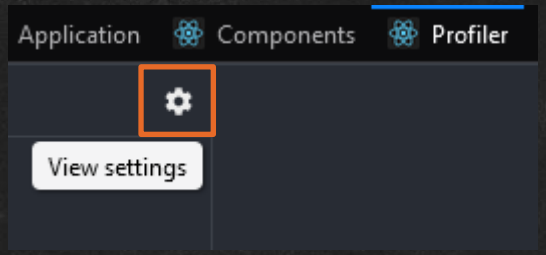
VS

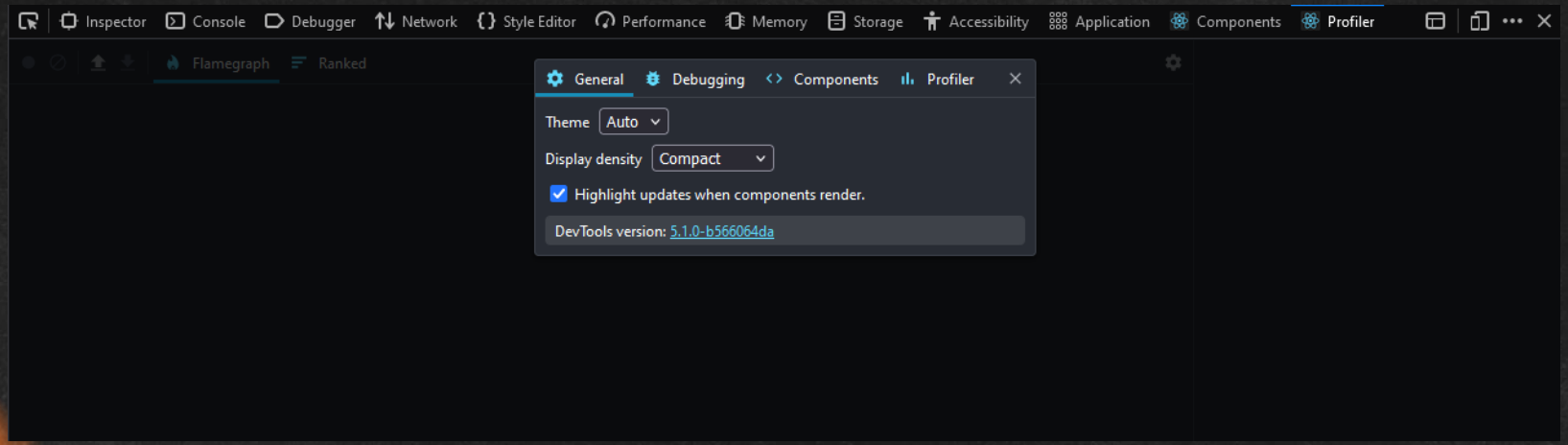
```
    <div>
      <h1># of rules broken</h1>
      <strong>{0}</strong>
      <button onClick={handleClick}>Increment</button>
    </div>
```


App
MyCounter
state: count (4)
Increment
Title
CounterDisplay
props: count

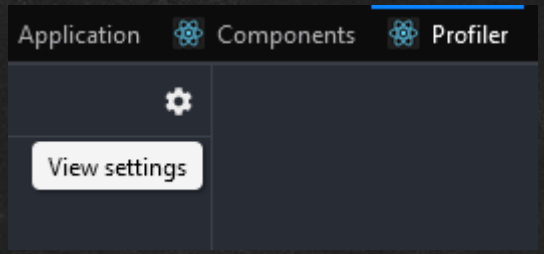

App
Increment
MyCounter
state: count (4)
React.memo()
Title
React.memo()
CounterDisplay
props: count

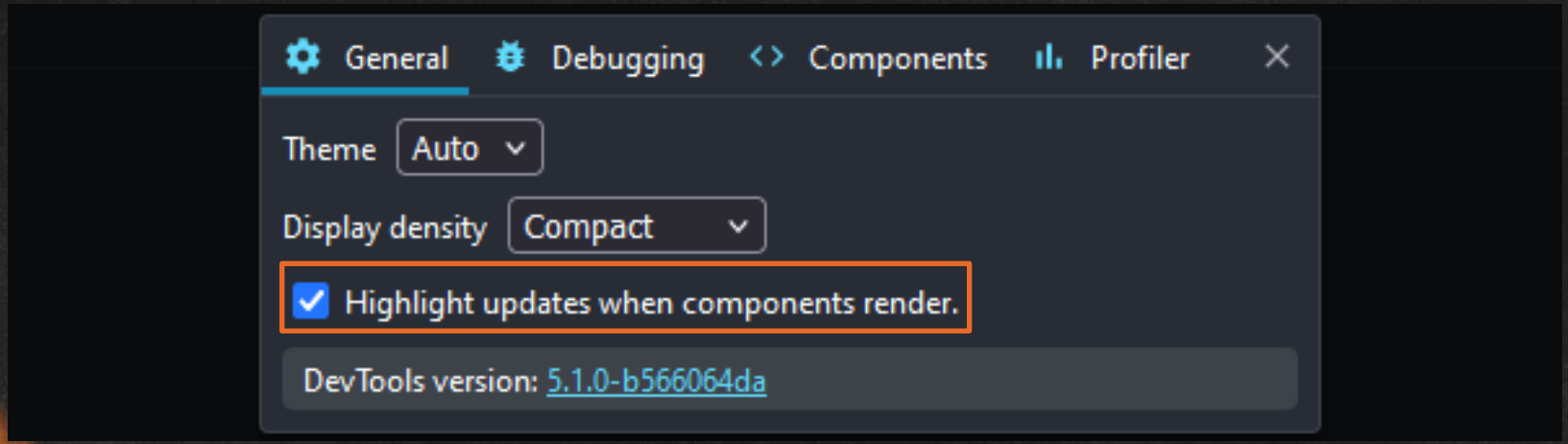
React Profiler: Highlight renders



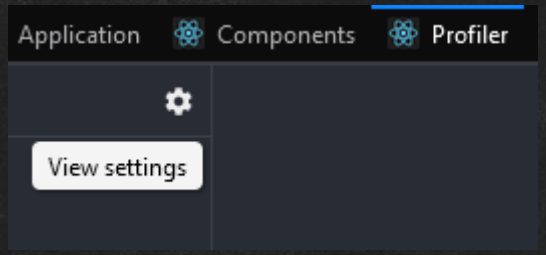


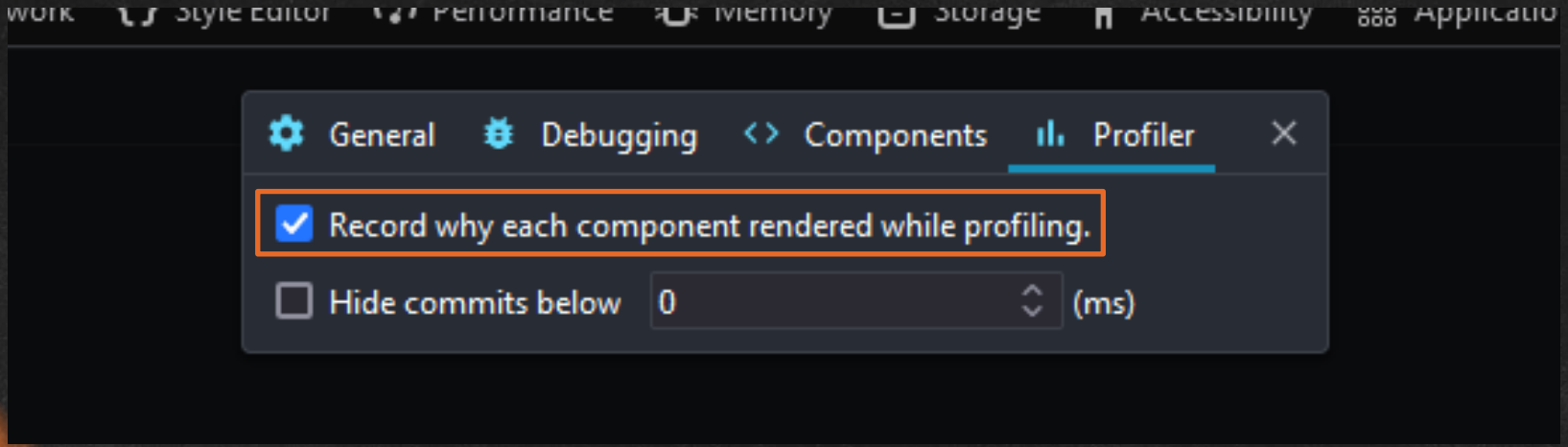
React Profiler: Highlight renders





React Profiler: Why did it render?



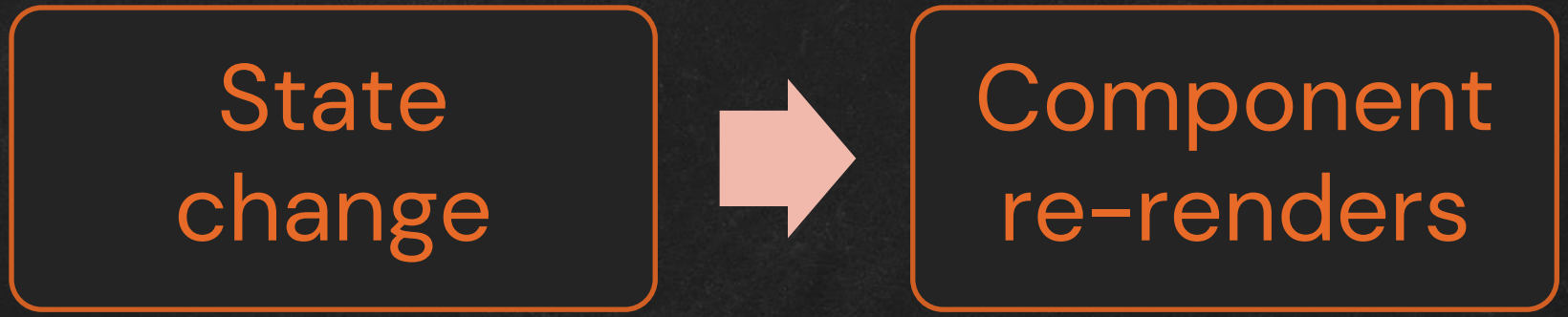


Why React Re-Renders

https://www.joshwcomeau.com/react/why-react-re-renders/

There are a lot of misconceptions out there about this topic, and it can lead to a lot of uncertainty. If we don't understand React's render cycle, how can we understand how to use `React.memo`, or when we should wrap our functions in `useCallback`??

Just always write performant components!



Every rerender starts with a state change.

02

Isolating expensive hooks

Isolate state changes with side effect components

Photo by Sam Pearce-Warrilow on Unsplash

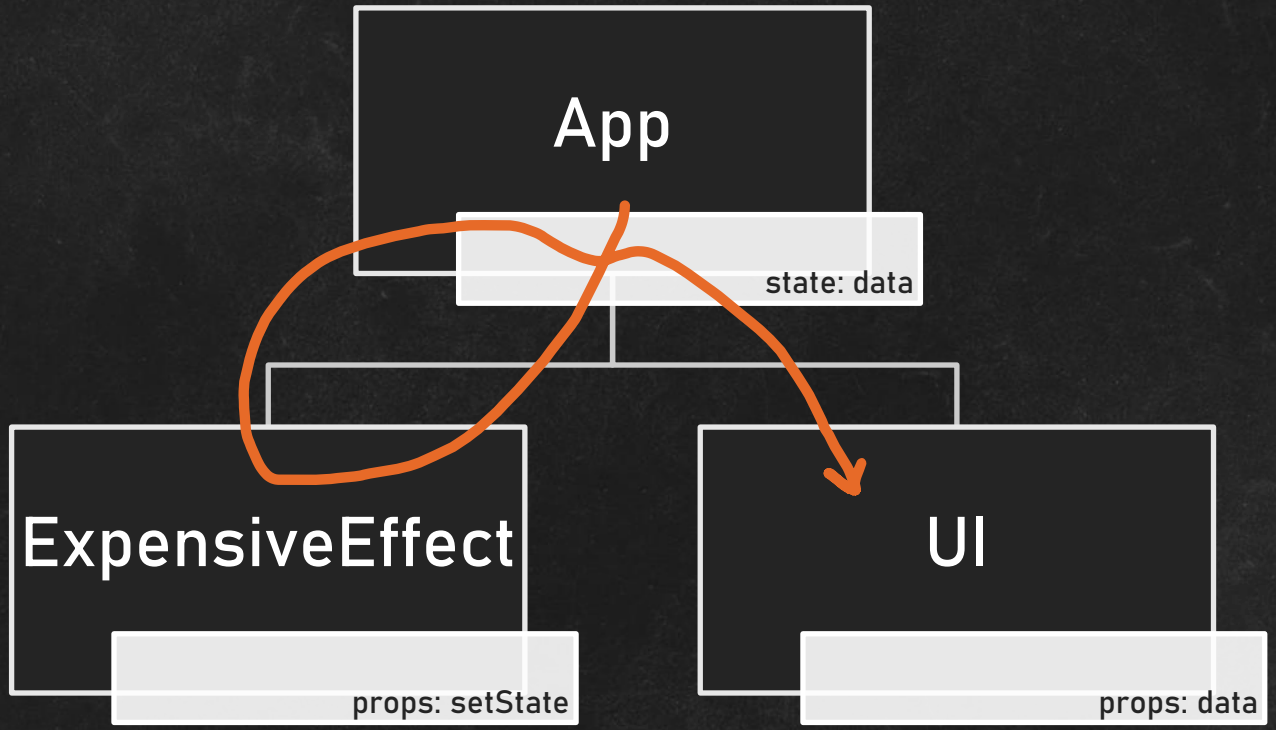
```
function useGetData() {
  const [state, setState] = useState();

  useEffect(() => {
    fetchData().then(setState);
  }, []);

  return state;
}
```

```
function App() {
  const data = useGetData();
  return <UI data={data} />;
}
```

```
function useGetData() {
  const [state, setState] = useState();
  const [otherState, setOtherState] = useState();

  useEffect(() => {
    fetchData().then(setState);
  }, []);

  useEffect(() => {
    fetchOtherData(state.id).then(setOtherState);
  }, [state.id]);

  return otherState;
}

function App() {
  const data = useGetData();
  return <UI data={data} />;
}
```

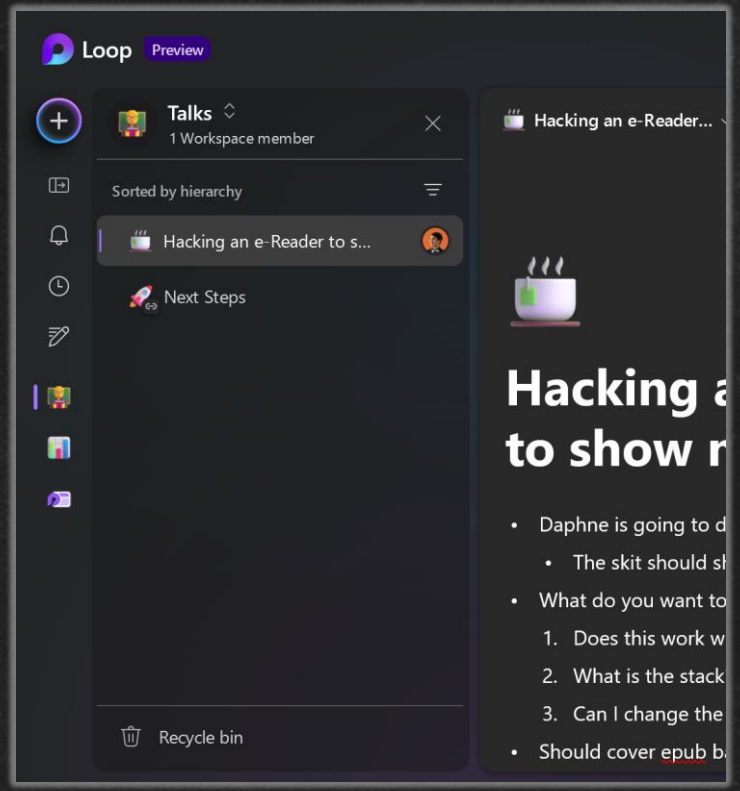
```
function useGetData() {
  const [otherState, setOtherState] = useState();

  useEffect(() => {
    fetchData()
      .then(state => fetchOtherData(state.id))
      .then(setOtherState)
  }, []);

  return otherState;
}



function App() {
  const data = useGetData();
  return <UI data={data} />;
}
```

```
function ExpensiveEffect(props) {
  const otherState = useGetData();

  useEffect(() => {
    props.setState(otherState);
  }, [otherState]);

  return null;
}



function App() {
  const [data, setData] = useState();
  return <>
    <ExpensiveEffect setState={setData} />
    <UI data={data} />
  </>;
}
```


App
state: data
ExpensiveEffect
props: setState
UI
props: data

5x speedup

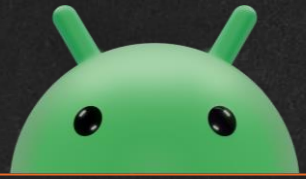
Refactoring in Microsoft Loop



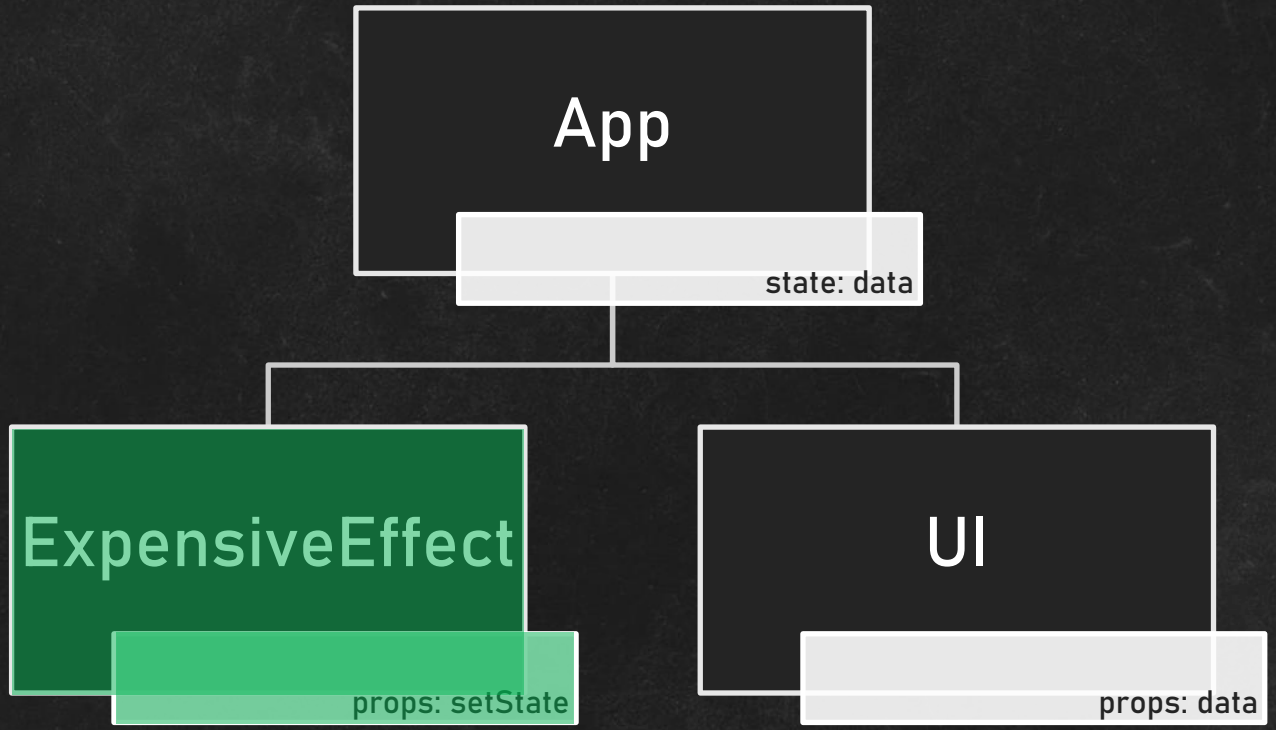
1x per state change

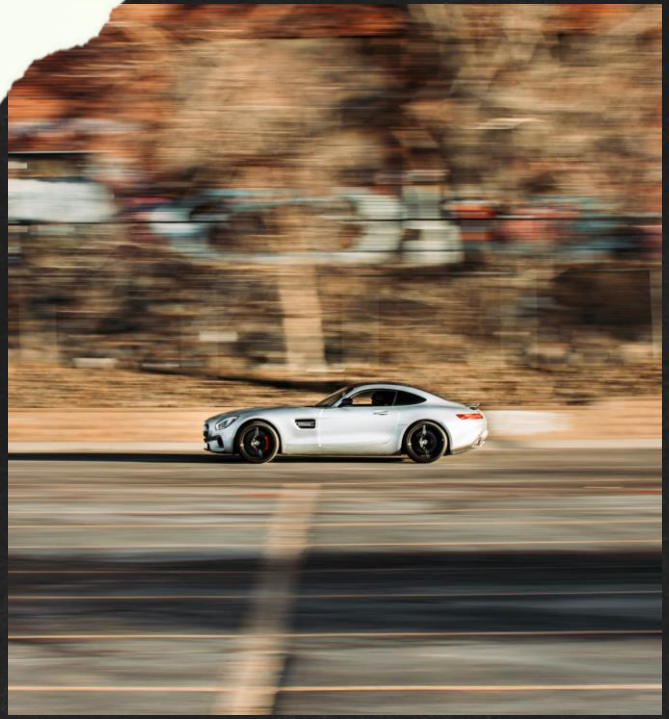
```
return (
  <>
    {shouldLoad && (
      <ExpensiveEffect setState={setData} />
    )}
    <UI data={data} />
  </>
);
```

```
const LazyExpensiveEffect = React.lazy(() =>
  import('./ExpensiveEffect’)
);


return (
  <>
    <React.Suspense fallback={null}>
      <ExpensiveEffect setState={setData} />
    </React.Suspense>
    <UI data={data} />
  </>
);
```



```
@Composable
fun loadData() {
  val (state, setState) = remember { mutableStateOf() }

  SideEffect {
    fetchData().then(::setState)
  }

  return state
}

@Composable
fun App() { … }
```


App
state: data
ExpensiveEffect
props: setState
UI
props: data

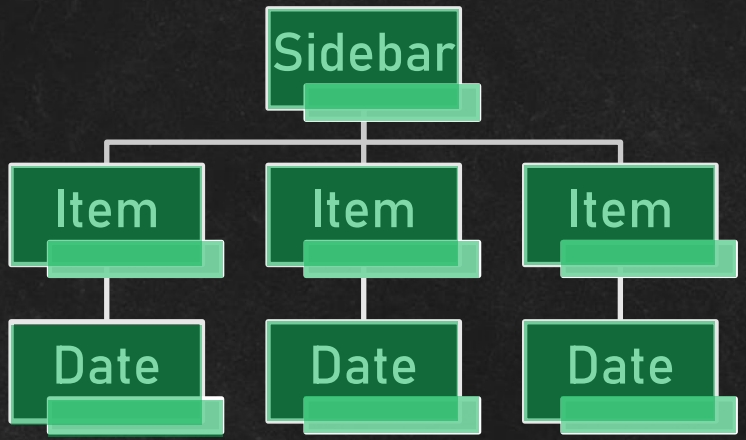


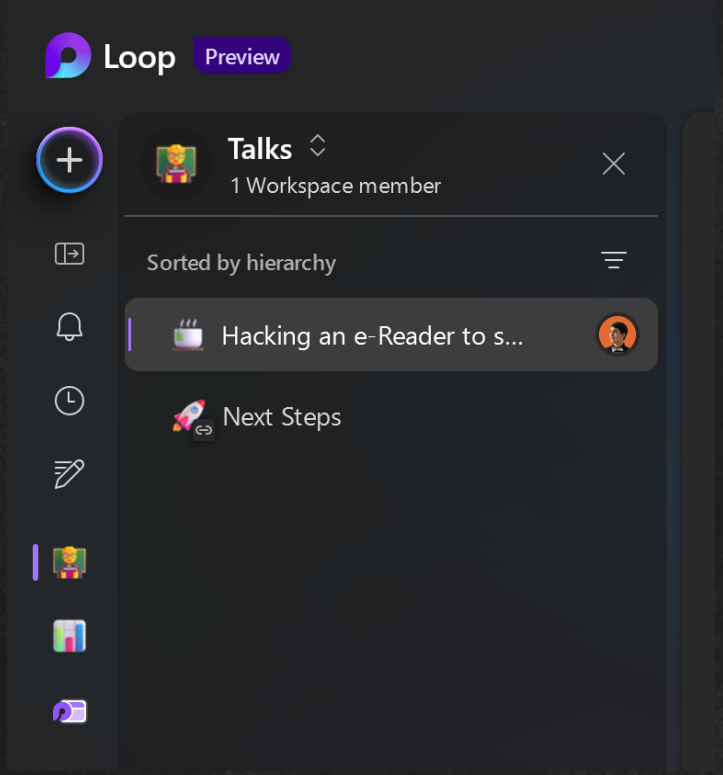
03

Only updating leaf components

Pull state management out of React

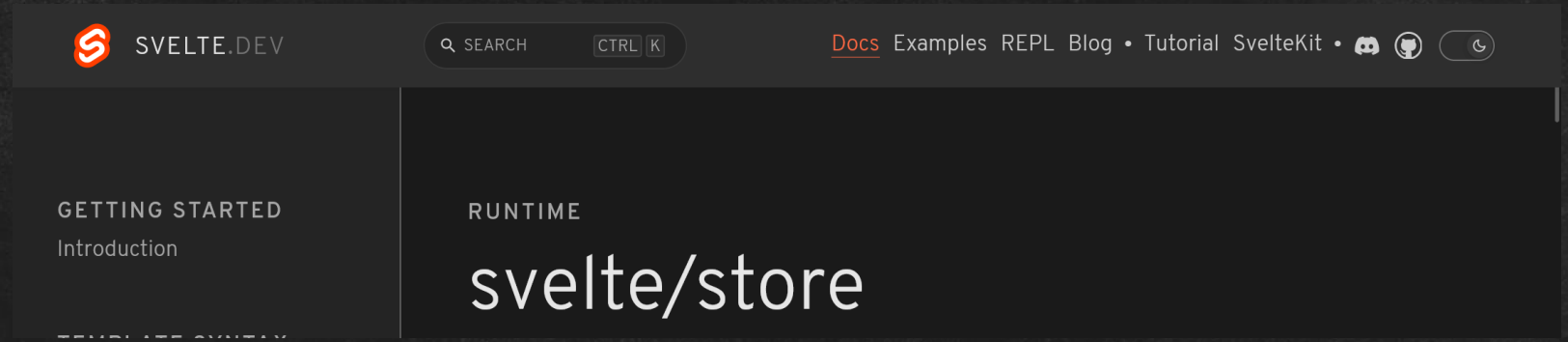
Photo by Jakob Rosen on Unsplash


Sidebar
Item
Item
Item
Date
Date
Date


Loop
Preview
Talks
1 Workspace member
Sorted by hierarchy
Hacking an e-Reader to s...
Next Steps

~~useState~~

Svelte



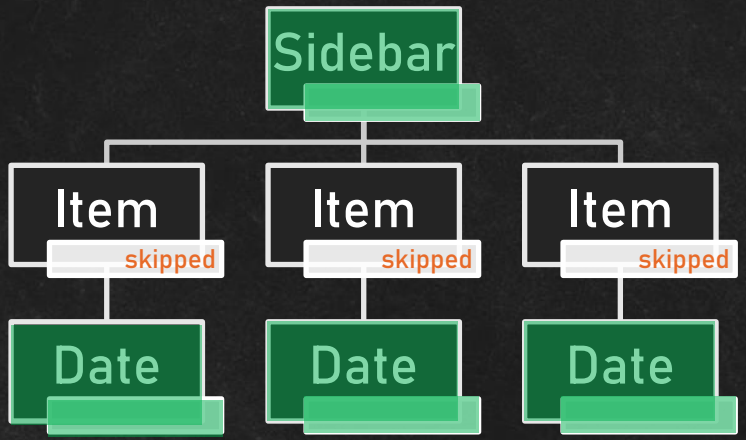

```
import { writable } from 'svelte/store';

let store = writable(initialState);

store.set(newState);
store.update(oldState => newState);
```

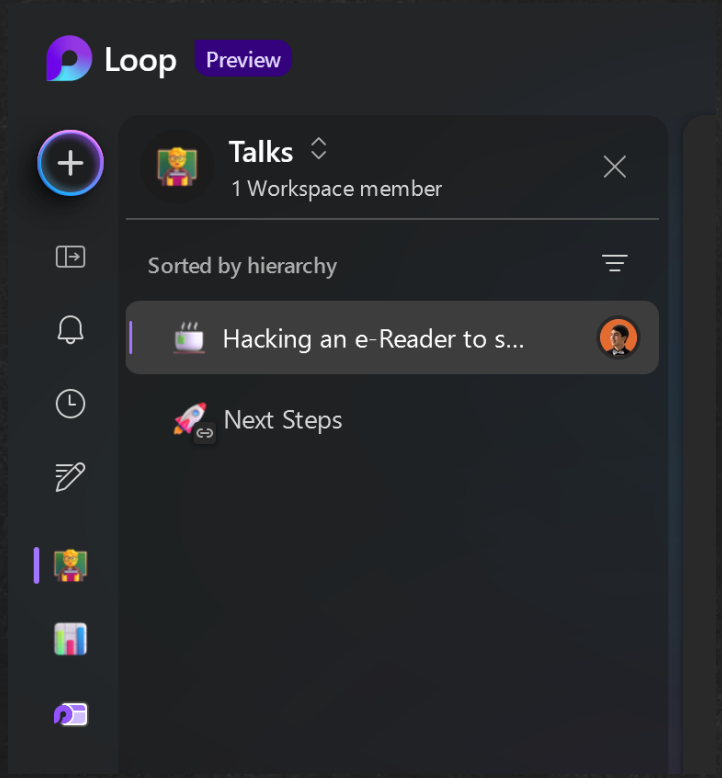
```
import {
  writable,
  derived,
} from 'svelte/store';

import {} from 'svelte';
```

```
import { writable } from 'svelte/store';

function Sidebar(props) {
  const store = React.useMemo(() => writable());

  return <>
    {props.items.map(item =>
      <Item item={item} state={store} />
    )}
  </>
}
```

```
import { writable } from 'svelte/store';

let store = writable(0);
store.set(1);
store === store;
```

```
import { useSyncExternalStore } from 'react';
import { get } from 'svelte/store';

// useSyncExternalStore internally uses
// useState and useEffect
const state = useSyncExternalStore(
  store.subscribe,
  () => get(store)
);
```


Sidebar
Item
skipped
Item
skipped
Item
skipped
Date
Date
Date


Loop
Preview
Talks
1 Workspace member
Sorted by hierarchy
Hacking an e-Reader to s...
Next Steps

Svelte Stores

- Store is a constant reference passed around with props or context.
- Store's value can be changed without re-rendering React tree.
- Translated into React state using the `useSyncExternalStore` hook.

```
function Sidebar(props) {
  const store = React.useMemo(() => writable());

  return <>
    <ExpensiveEffect setState={store.set} />
    {props.items.map(item =>
      <Item item={item} state={store} />
    )}
  </>
}
```

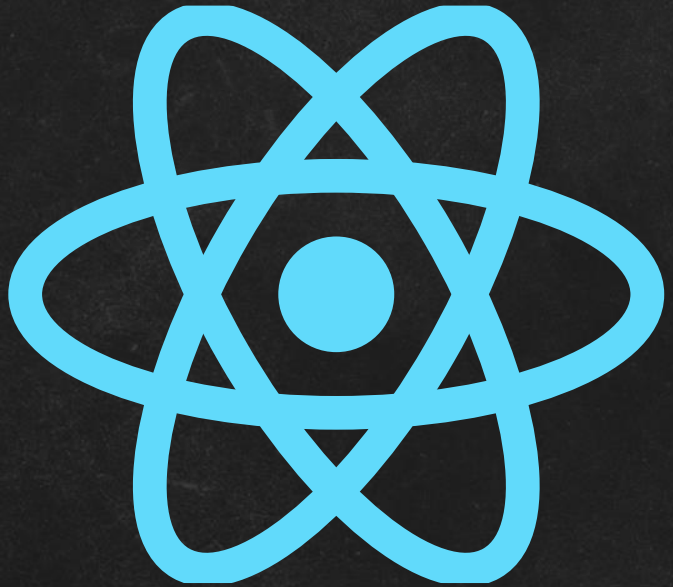
```
import { writable, readable } from 'svelte/store';

class DataModel {
  #state = writable();

  get state() {
    return readable(this.#state);
  }

  performUpdate() {
    this.#state.set('done')
  }
}
```

Redux

- Store is a constant reference passed around with `<Provider />`.
- Store's value can be changed without re-rendering React tree.
- Translated into React state using the `useSelector` hook.

React.useReducer

- Reducer value is stored as one giant React state object.
- Changing reducer's value will re-render React tree.
- State is translated at the top of the React tree once, instead of in the leaf components.



Could you use Signals?

Yes.

```
import { readable } from 'svelte/store';

const mediaQuery = window.matchMedia('(max-width: 640px)');
const mediaQueryStore = readable(
  mediaQuery.matches,
  function onFirstListenerStart(set) {
    const listener = () => set(mediaQuery.matches);
    listener();
    mediaQuery.addEventListener('change', listener);
    // return cleanup function, similar to useEffect
    return function onLastListenerStop() {
      mediaQuery.removeEventListener('change', listener);
    }
  }
);
```

tc39 / proposal-signals ☆ 2.7k ✓

Type / to search

Code | Issues 87 | Bugs 2 | Pull requests 10 | Actions | Security | Insights

proposal-signals

Public

generated from tc39/template-for-proposals

Watch 85 | Fork 50 | Star 2.7k

README | Code of conduct | MIT license | Security

JavaScript Signals standard proposal

Stage 1 (explanation)

TC39 proposal champions: Daniel Ehrenberg, Yehuda Katz, Jatin Ramanathan, Shay Lewis, Kristen Hewell Garrett, Dominic Gannaway, Preston Sego, Milo M, Rob Eisenberg

Original authors: Rob Eisenberg and Daniel Ehrenberg

This document describes an early common direction for signals in JavaScript, similar to the Promises/A+ effort which preceded the Promises standardized by TC39 in ES2015. Try it for yourself, using a polyfill.

Similarly to Promises/A+, this effort focuses on aligning the JavaScript ecosystem. If this alignment is successful, then a standard could emerge, based on that experience. Several framework authors are collaborating here on a common model which could back their reactivity core. The current draft is based on design input from the authors/maintainers of Angular, Bubble, Ember, FAST, MobX, Preact, Qwik, RxJS, Solid, Starbeam, Svelte, Vue, Wiz, and more...

Differently from Promises/A+, we're not trying to solve for a common developer-facing surface API, but rather the precise core semantics of the underlying signal graph. This proposal does include a fully concrete API, but the API is not targeted to most application developers. Instead, the signal API here is a better fit for frameworks to build on top

A proposal to add signals to JavaScript.

- Readme
- MIT license
- Code of conduct
- Security policy
- Activity
- Custom properties
- 2.7k stars
- 85 watching
- 50 forks
- 9 months old

Contributors 19

TypeScript 100.0%

THANKS!

https://loop.microsoft.com

tigeroakes.com

@notwoods.bsky.social

@not_woods

Photo by Fabio Spinelli on Unsplash