Understanding Render Optimization

In this lesson, we will learn what render optimization is and when we need to use it.

WE'LL COVER THE FOLLOWING

Visualizing renders

Before explaining render optimization, we must modify our counter app so that we can visualize renders.

```
import { useRef, useEffect } from 'react';

const useFlasher = () => {
  const ref = useRef(null);
  useEffect(() => {
    ref.current.classList.add('flash');
    setTimeout(() => {
      ref.current.classList.remove('flash');
    }, 300);
  });
  return ref;
};
```

This adds `flash` class to a DOM that the hook is attached to.

We assume that `flash` class style is defined in the css.

```
const Counter = ({ name }) => {
  const [state, dispatch] = useGlobalState();

  return (
    <div ref={useFlasher()}>
      {state[name]}
      <button onClick={() => dispatch({ type: 'INCREMENT', name })}>+1</bu
tton>
      <button onClick={() => dispatch({ type: 'DECREMENT', name })}>-1</bu
tton>
    </div>
  );
};
```

The counter component from the previous example is modified just a bit. The only change is `ref={useFlasher()}`.

Check the app

Now, let's see how it behaves in the following app.

```
import React from 'react';
require('./style.css');

import ReactDOM from 'react-dom';
import App from './app.js';

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Render optimization

The app above should behave in a way that all components (the 4 numbers) render if any button is clicked. However, **Count2** components wouldn't need to re-render when only `count1` is incremented or decremented, and vice versa.

Render optimization means only those components re-render that display a change made to the previous version. This is important for performance in certain cases.

The primary approach to render optimization

In most cases, render optimization is not necessary as long as the app works.

In some intensive apps, performance is critical.

The primary approach in such apps is to split a context into multiple contexts. In our counter app, we can create two contexts for two counts.

Render optimization by React Tracked

For certain reasons, we might not want to split a context; maybe because the state is tightly coupled and it makes sense to update the state at once.

React Tracked solves this issue by state usage tracking. State usage tracking is to track the usage of the state in render and only trigger re-renders if the used part of the state is changed. In other words, if you click “+1” in Count1, only Count1 components re-render, and Count2 components will not re-render.

Next

We will see how render optimization behaves with React Tracked in the next lesson.