

Practical Performance Optimization in React

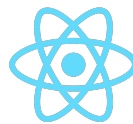
Kemet Dugue



kemetdugue




kdugue



Agenda

1. Brief Introduction
2. Sensible Optimization & Popular Use Cases
 - a. Unnecessary Re-rendering & Expensive Computations
 - b. Lists & Virtualization
3. Tools for Debugging (React Dev Tools)
4. Performance Best Practices
5. Q+A

About Me

- Engineer at 
- Interests:
 - lo-fi house
 - tennis
 - cats



Whatever brings you joy!

Opinions and Q+A

Sensible Optimization aka When is the *right* time to optimize?

- App functionality slowness
 - Scrolling
 - Updating of views
 - Click events
- React component design (use best practices)
- Don't prematurely optimize!



Popular Optimization Use Cases

- Unnecessary re-renders
- Expensive Computations
- Collections (small and large)
- Large App Bundle*



Component to DOM (Two Phases)

1. Render phase

- JSX transformed
- Can be called multiple times for a given change (**re-render**)
 - i. Component state change
 - ii. Props Change
 - iii. (in case of children) Parent re-renders
- Reconciliation:
 - i. “Diffing” algorithm: determines parts of the node tree that have to be updated
 - ii. Current virtual DOM to previous virtual DOM

2. Commit phase

- changes to actual DOM (**commits** to changes resulted from reconciliation);
- Called only once for a given change

```
const MyProfile = (props) => {  
  const { profile, isReady, actions } = props;  
  
  <ProfileBanner  
    shouldShow={isReady}  
    name={profile.name}  
    profileBannerClick={() => actions.openModal(true)}  
  />;  
};
```



```
· · · <LoadingBar
· · ·   · scope={customer.id}
· · ·   · style={{
· · ·     · backgroundColor: "black",
· · ·     · height: "3px",
· · ·     · marginTop: "-9px",
· · ·     · marginLeft: "7px",
· · ·   · }}
· · ·   · fullWidth
· · ·   · small
· · ·   · text="Please wait wi"
· · · />
```

- Assume props are rarely changing

```
const ProfilePage = ({ name, instagramUrl, profileName }) => {  
  return (  
    <div>  
      <ProfileBanner bannerTitle={profileName} />  
      <ProfilePageHeader name={name} instagramUrl={instagramUrl} />  
    </div>  
  );  
};  
  
const ProfileBanner = ({ bannerTitle }) => {  
  return <div className="profile-banner">{bannerTitle}</div>;  
};  
  
const ProfilePageHeader = ({ name, instagramUrl }) => {  
  return (  
    <div>  
      <p> Welcome back, {name}! </p>  
      <a href={instagramUrl}>Your Instagram</a>  
    </div>  
  );  
};
```

```
const ProfileBanner = React.memo(({ bannerTitle }) => {  
  ··return <div className="profile-banner">{bannerTitle}</div>;  
});
```

```
const ProfilePageHeader = React.memo(({ name, instagramUrl }) => {  
  ··return (  
    ··<div>  
    ··<p> Welcome back, {name}! </p>  
    ··<a href={instagramUrl}>Your Instagram</a>  
    ··</div>  
  ··);  
});
```

Memoize Expensive Calculations/ Components

- Use for Expensive calculations
- React.memo for functional components
- useMemo to memoize values
- PureComponent for class components

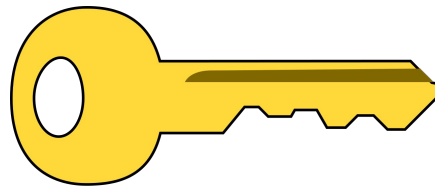
```
const GraderScore = ({ category }) => {  
  // some other logic  
  // very expensive calculation  
  const grade = calculateGrade(category);  
  return <div>{grade}</div>;  
};
```



```
const GraderScore = ({ category }) => {  
  // some other logic  
  const memoizedGrade = useMemo(() => calculateGrade(category), [category]);  
  return <div>{memoizedGrade}</div>;  
};
```

Rendering Collections

- Using index as key can be sometimes deceptive
- Still, index as key can negatively affect performance



Add Item to the beginning of the list

key=index

Item# 1444610101010

value for 1444610101010

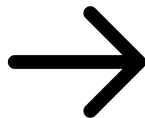
Item# 1444600000000

key=id

Item# 1444610101010

value for 1444610101010

Item# 1444600000000



Add Item to the beginning of the list

key=index

Item# 1619990662001

value for 1444610101010

Item# 1444610101010

Item# 1444600000000

key=id

Item# 1619990662001

Item# 1444610101010

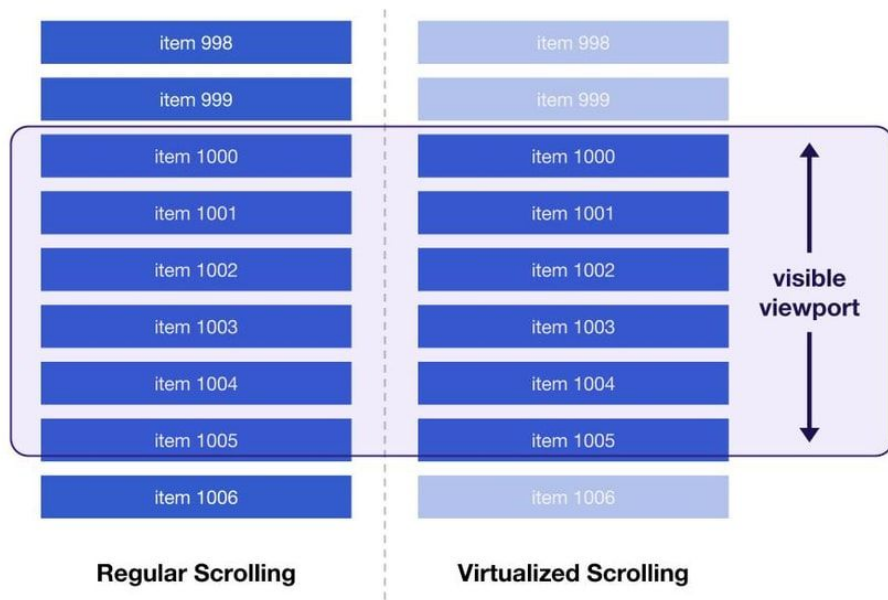
value for 1444610101010

Item# 1444600000000

Credit: Robin Pokorny

Large Collection Solution: List Virtualization (Windowing)

- Rendering 1000's of rows
- Virtualization: rendering only what is in the viewport
- Popular Libraries:
 - react-window
 - react-virtualized
- Tradeoff: pay in rendering time later rather than upfront (initial load)



Credit: web.dev

Todo App with React Dev Tools!

TodoInput

TodoList

feed Chucky

finish presentation [Todo](#)

get groceries

listen to podcast

clean

have fun!

Best Practices

- Make sure optimization is a sensible one!
 - Every optimization comes with a tradeoff
 - Measure performance first
- Don't pass new references (arrays, objects, functions) as props to components.
- Use a unique and stable key value for a list. Minimum usage is index (be cautious)
- Pass only the props that are needed by the component (no "...props")
- Consider PureComponent (class) or React.memo (functional) for the following scenarios:
props that rarely change, components that renders often
- useMemo to memoize expensive calculations

Best Practices (cont'd)

- useCallback* to maintain callback reference
 - EX: child component is already memoized and takes in that callback as prop
- Use React Dev Tools for debugging components.
- Don't query DOM directly (EX: document.querySelector)
- Keep constructor() light

Resources

- React official docs: <https://reactjs.org/>
- React Dev Tools Tutorial: <https://react-devtools-tutorial.vercel.app/>
- List Virtualization Libraries
 - react-window: <https://github.com/bvaughn/react-window>
 - react-virtualized: <https://github.com/bvaughn/react-virtualized>
- Code-Splitting (for large app bundles):
<https://reactjs.org/docs/code-splitting.html>

Questions????