Improving Component UI Performance

Peter Kellner

Developer and Author

peterkellner.net | linkedin.com/in/peterkellner99 Mastodon: @pkellner@peterkellner.net

React Without Optimization

React efficiently manipulates the browser DOM

React is optimized around expected browser workloads

Most Common Optimizations in React

Fixing over-rendering

Using the React Dev Tools Flamegraph to find over-rendering problems

Solving over-rendering problems with React.memo

Prioritizing state updates

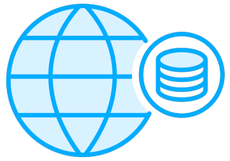
Long running tasks may need optimization

Prioritizing state updates with React API calls useDeferredValue and useTransition

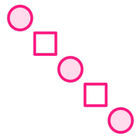
When which API call to use based on the problem

Fixing over Rendering in React Apps

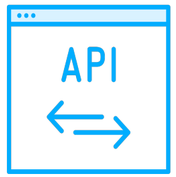
Fixing over Rendering in React Apps



React apps “React” based on local component state changes



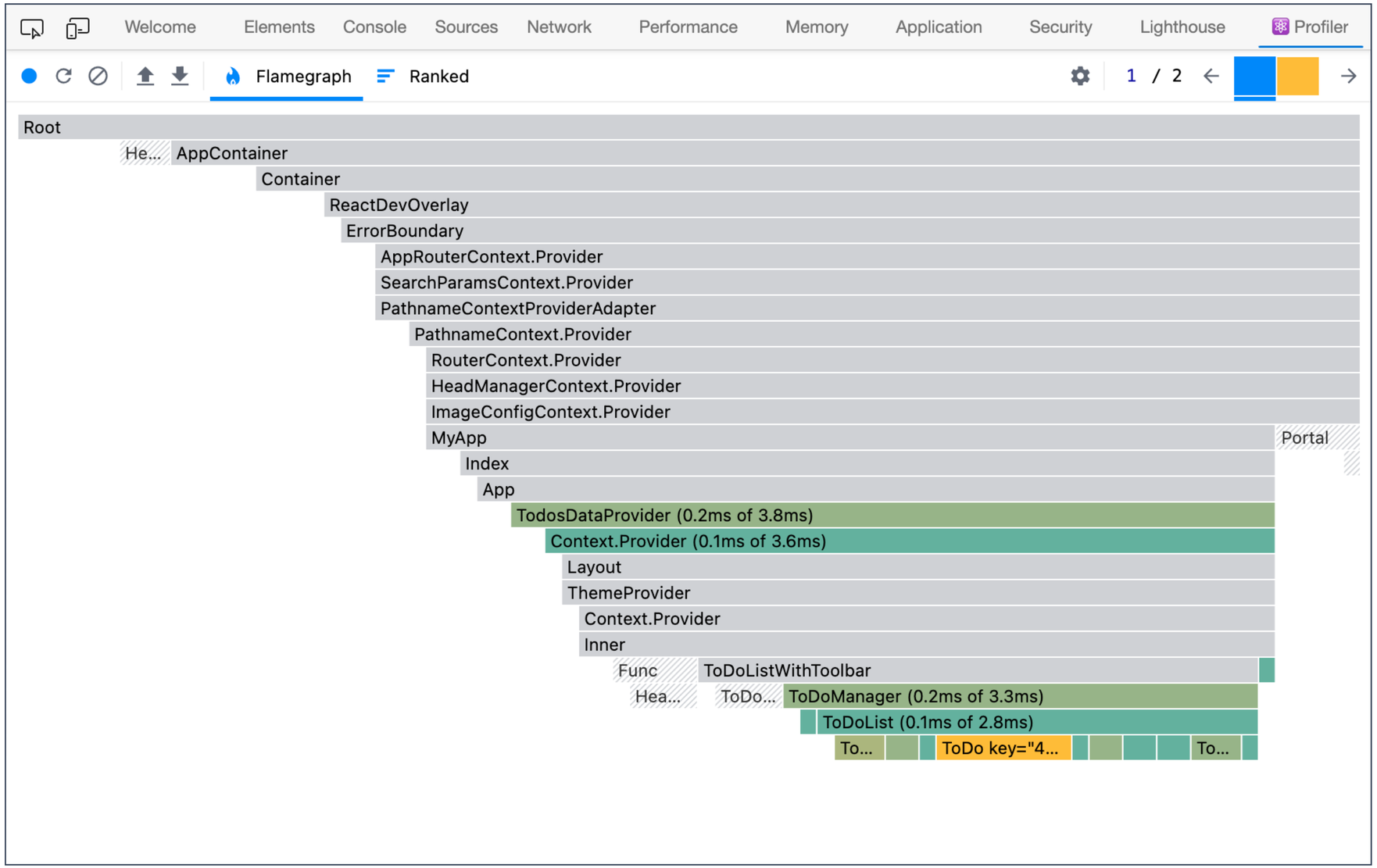
React is efficient at updating the browser DOM



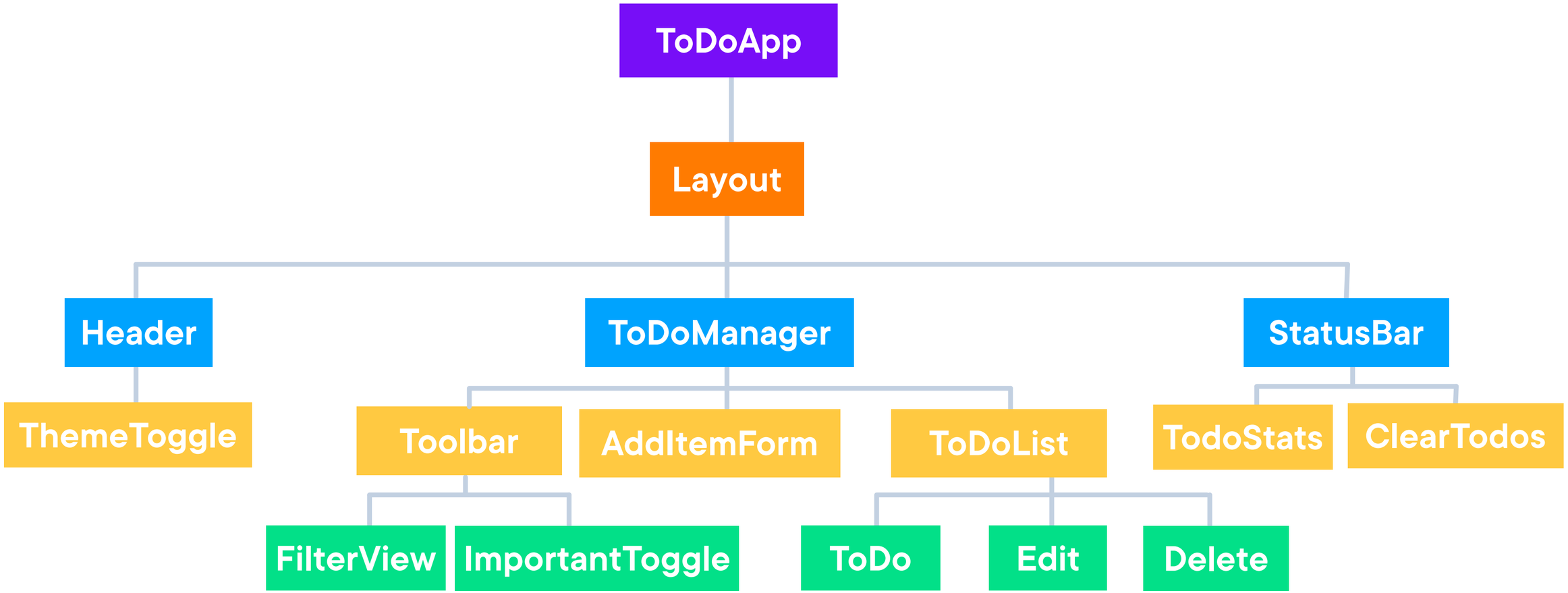
React API provides calls to help reduce over rendering

React.memo is most common API call for reducing over rendering

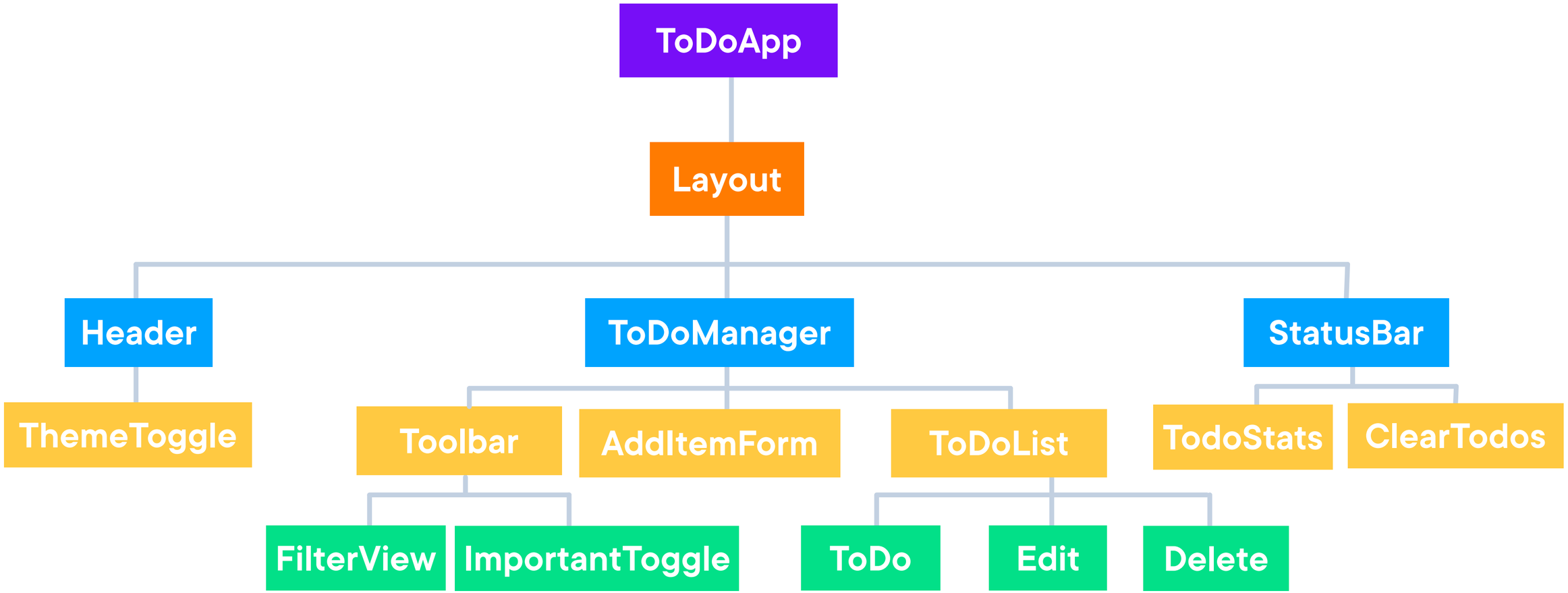
React Component Hierarchy



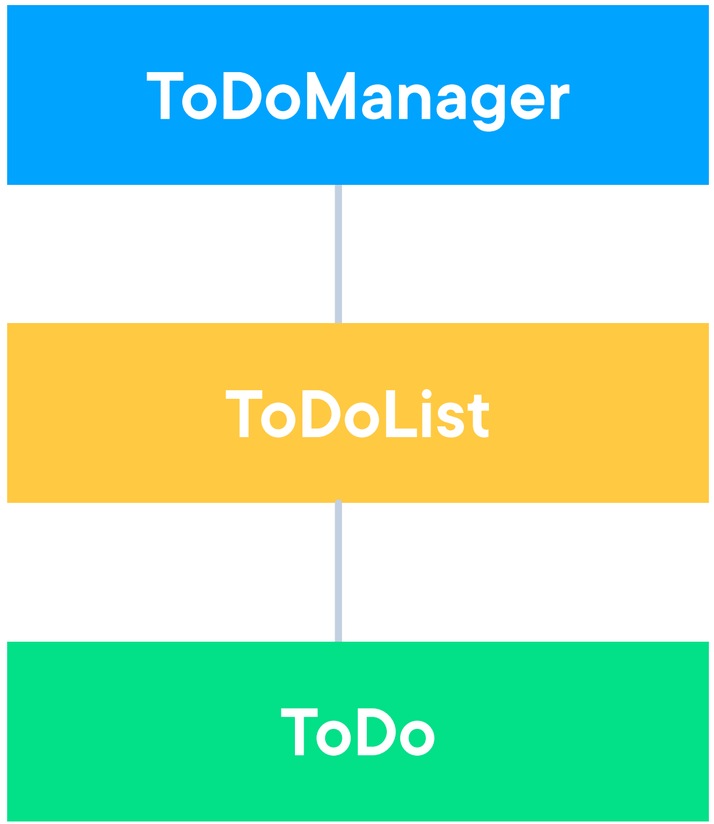
React Component Hierarchy



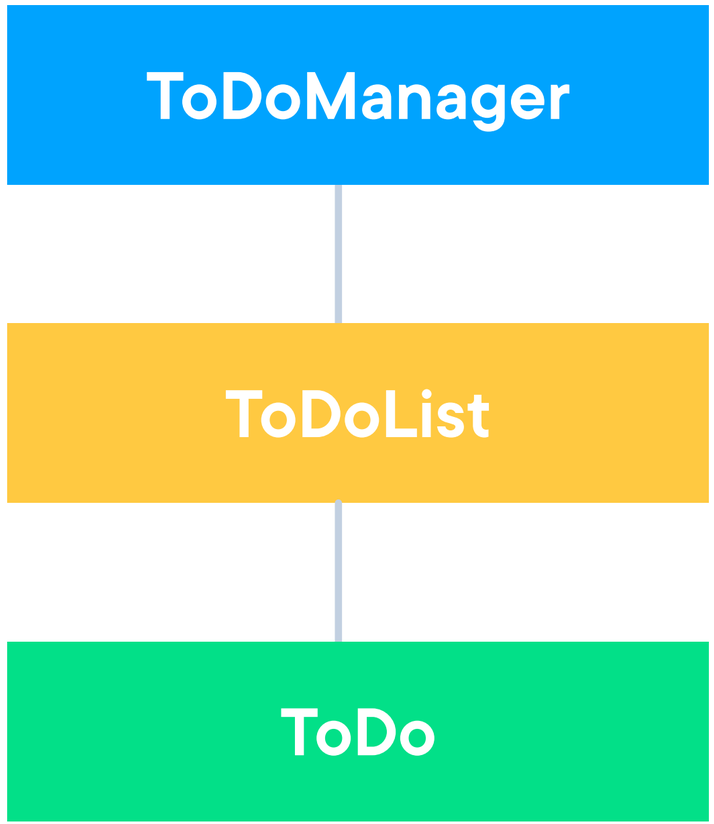
React Component Hierarchy



React Component Hierarchy

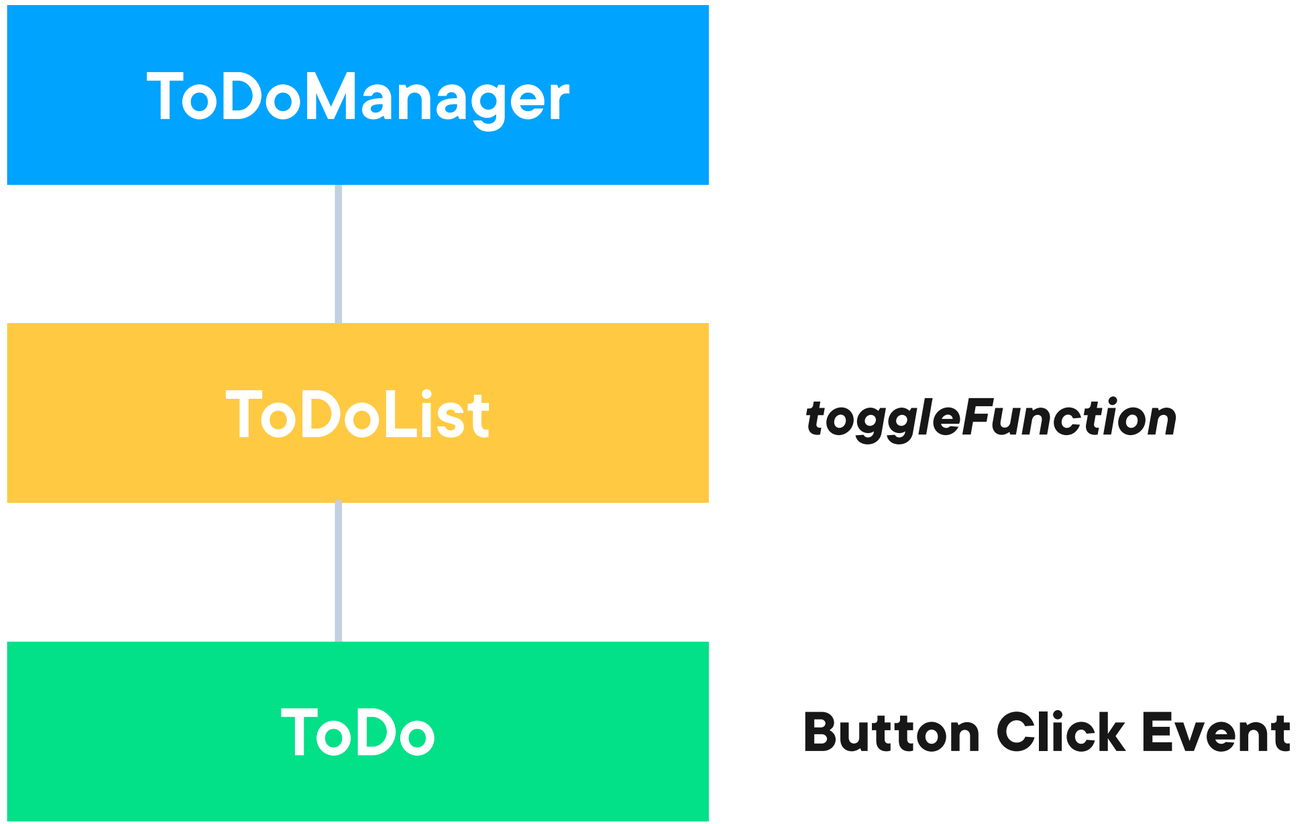


React Component Hierarchy

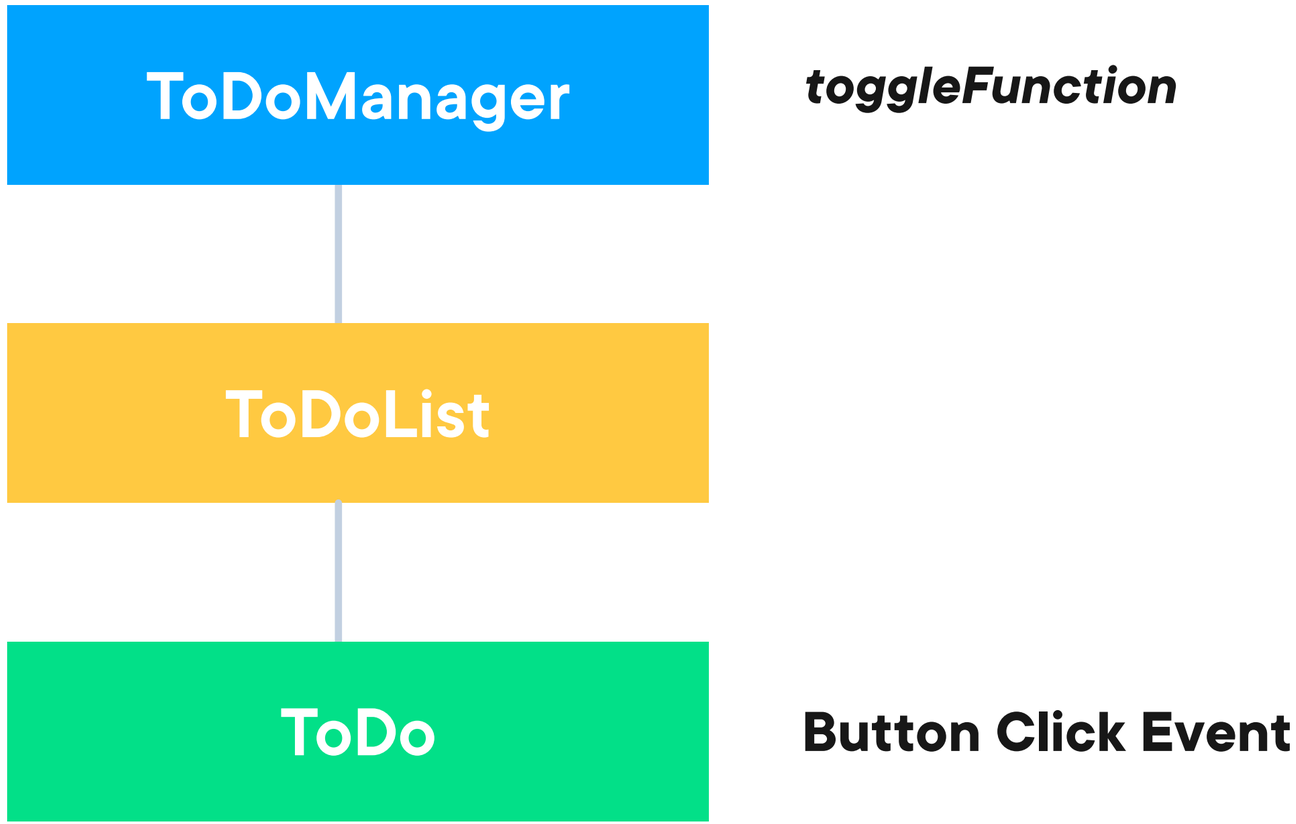


Button Click Event

React Component Hierarchy



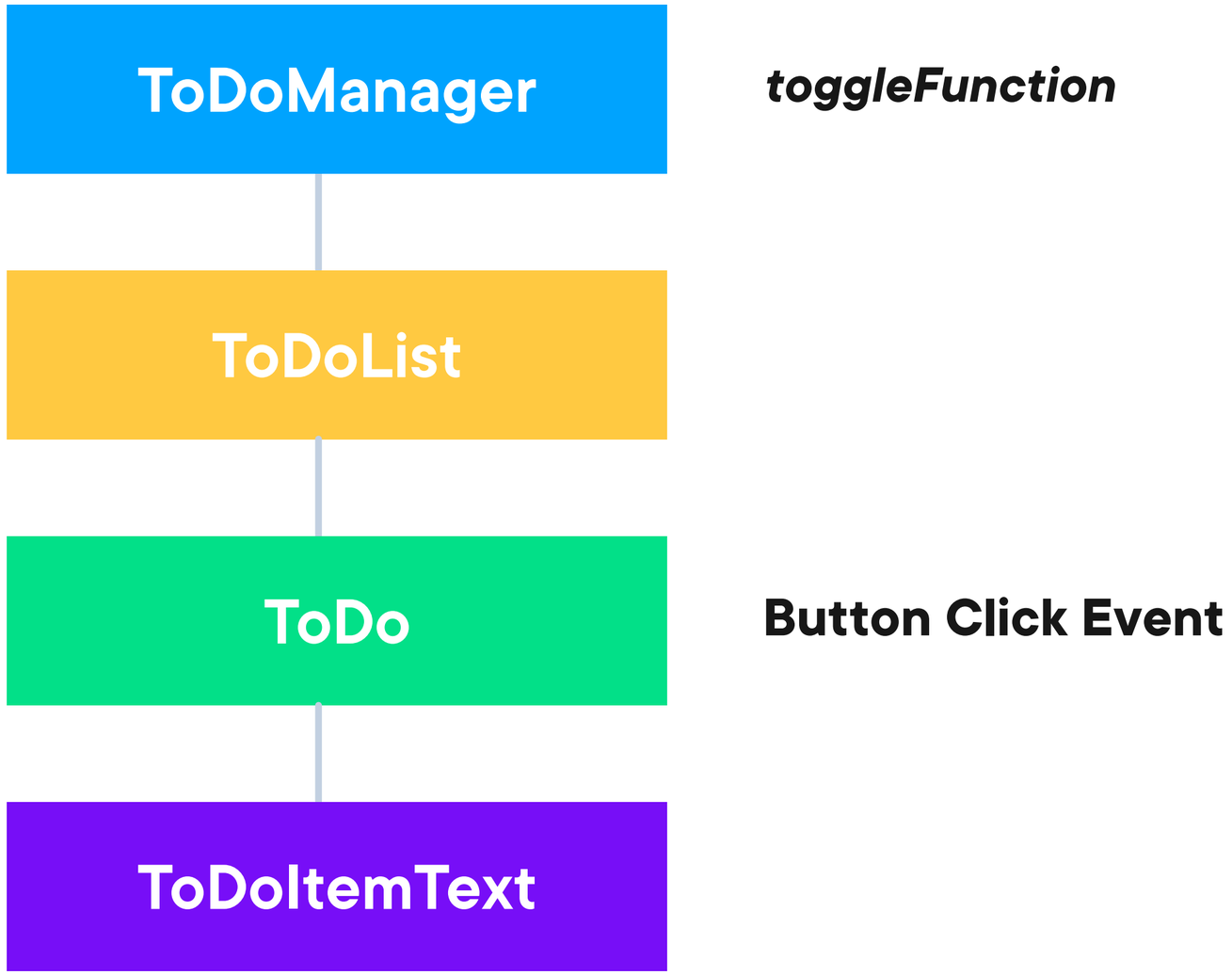
React Component Hierarchy



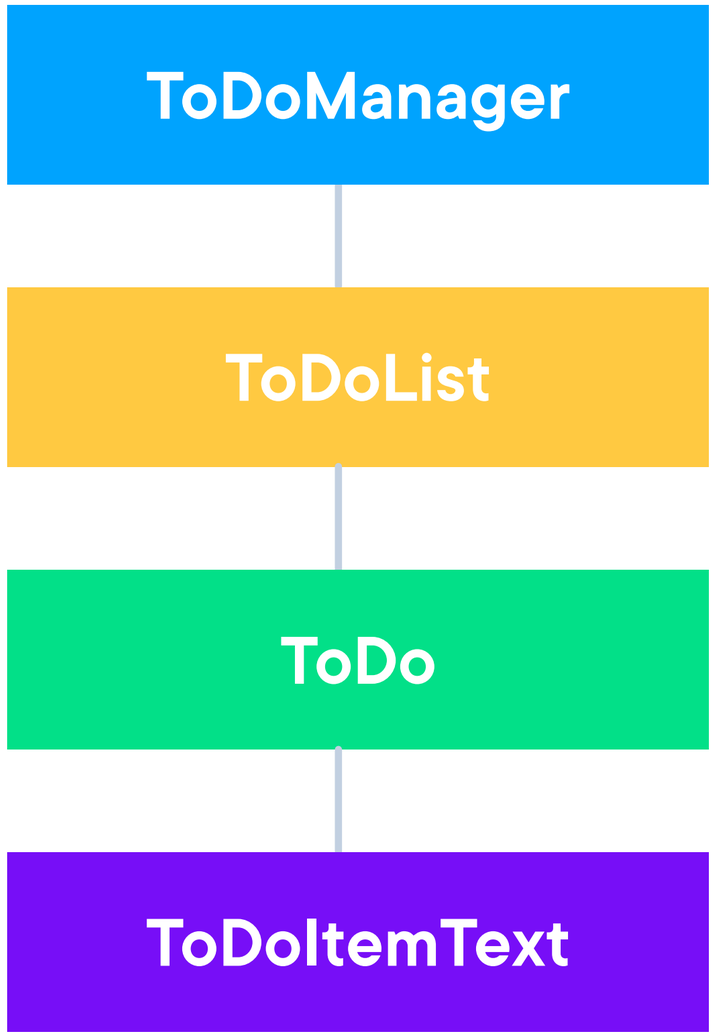
React.memo Can Solve over Rendering Problems

What happens when a child component is called by a parent component?

React Component Hierarchy



React Component Hierarchy



Why Parent Components Cause Children to Re-render

```
function MyComponent({ myNumber }) {

  return (
    <div>
      {myNumber}
    </div>
  );
}
```

Why Parent Components Cause Children to Re-render

```
function MyComponent({ myNumber}) {
  const displayDate = new Date().toLocaleDateString();
  return (
    <div>
      {myNumber}
    </div>
  );
}
```

Why Parent Components Cause Children to Re-render

```
function MyComponent({ myNumber}) {
  const displayDate = new Date().toLocaleDateString();
  return (
    <div>
      {myNumber} - {displayDate}
    </div>
  );
}
```

Implementing React.memo to Solve Our Re-rendering Problem

What Does React.memo Do?

```
React.memo()
```

What Does React.memo Do?

```
React.memo(originalComponent)
```

What Does React.memo Do?

```
const memoizedComponent =
  React.memo(originalComponent)
```

Advanced Usage of React.memo with a Custom Function

What Does React.memo Do?

```
function ToDoItemText({important,todoText }) {
  return <>...</>
}
```

React.memo Advanced Usage

```
const memoizedComponent = React.memo(
  originalComponent,
);
```

React.memo Advanced Usage

```
const memoizedComponent = React.memo(
  originalComponent,
  () => {}
);
```

React.memo Advanced Usage

```
const memoizedComponent = React.memo(
  originalComponent,
  (prevProps, nextProps) => {}
);
```

React.memo Advanced Usage

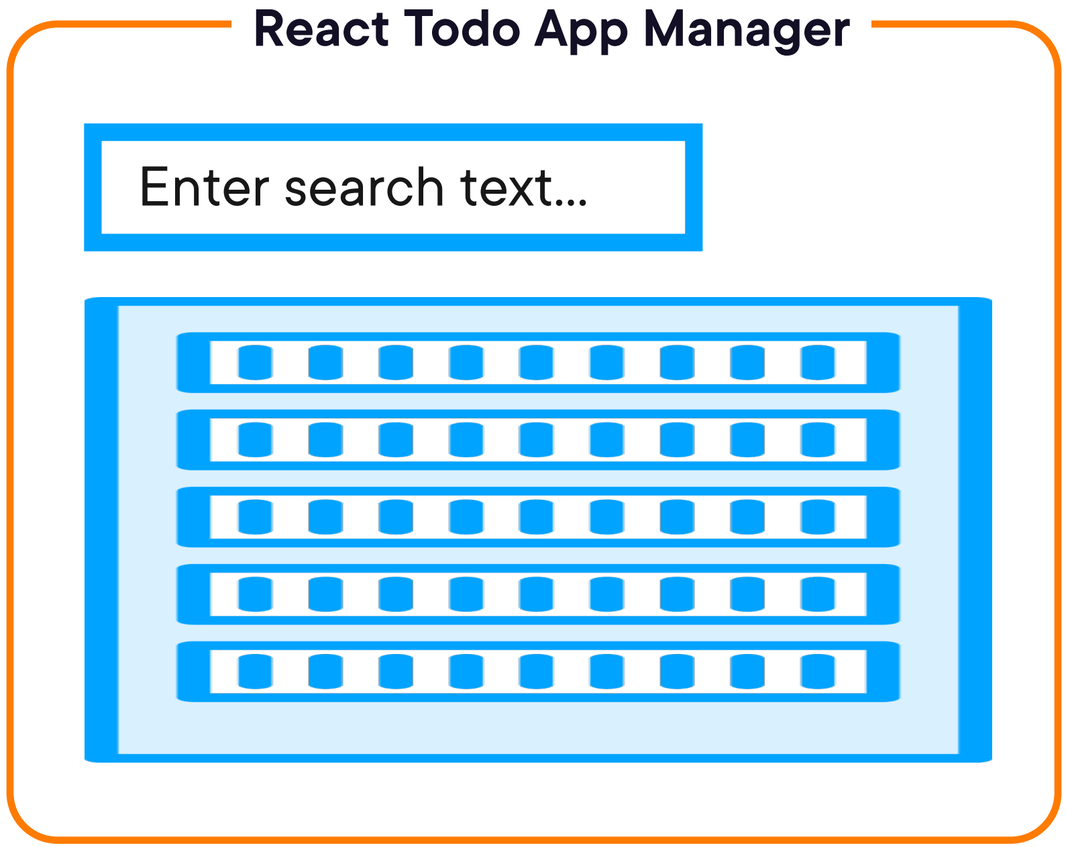
```
const memoizedComponent = React.memo(
  originalComponent,
  (prevProps, nextProps) => {
    return prevProps.todoItem.important != nextProps.todoItem.important ||
        prevProps.todoItem.itemText != nextProps.todoItem.itemText;
  }
);
```

What Does React.memo Do?

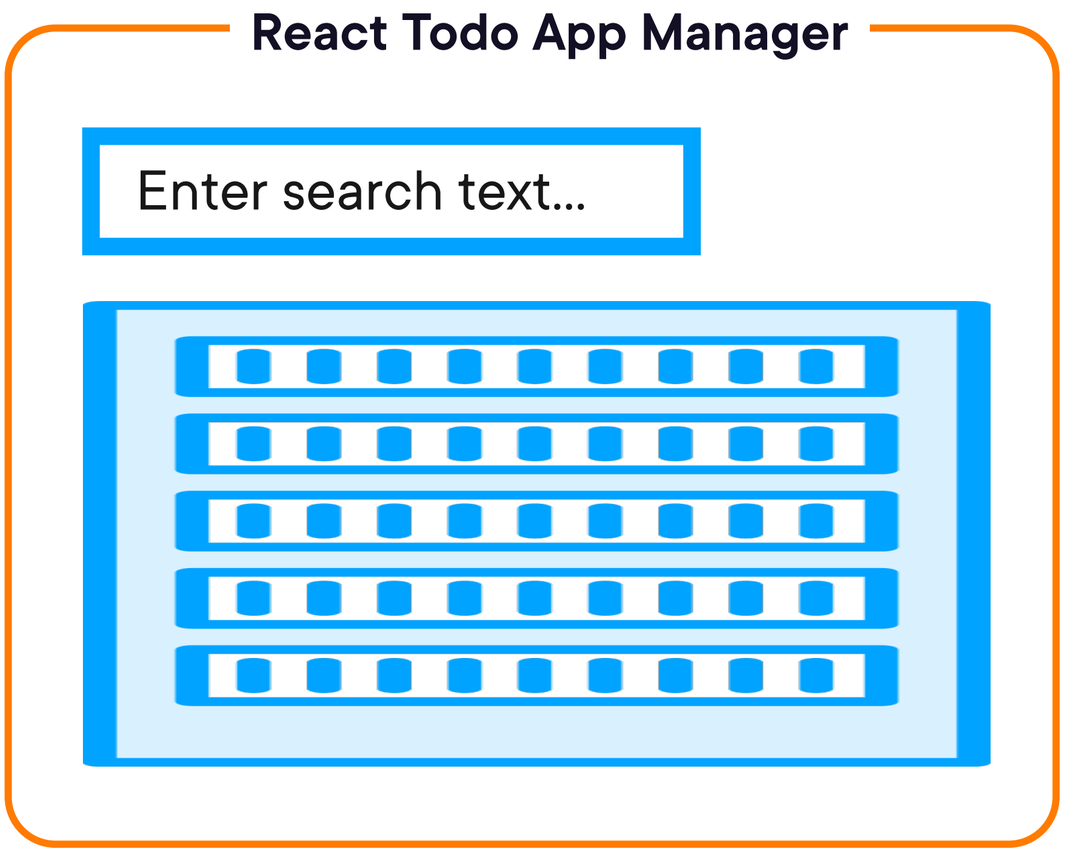
```
React.memo(originalComponent)
```

Deferring UI Updates with useDeferredValue

Typical React App (Todo Manager)



Typical React App (Todo Manager)



The React API includes a library call useDeferredValue to defer state value updates

Typical React App (Todo Manager)

```
function App() {
  const [search, setSearch] = useState("");

  const [todoList, setTodoList] = useState([
    "clean dog","eat lunch","wash clothes","..."]);

  return (
    <div>
      <input value={search}
         onChange={(e) => setSearch(e.target.value)} />
      <ShowTodoList list={todoList} search={search} />
    </div>
  );
}
```

Typical React App (Todo Manager)

```
function App() {
  const [search, setSearch] = useState("");
  const deferredSearch = useDeferredValue(search);

  const [todoList, setTodoList] = useState([
    "clean dog","eat lunch","wash clothes","..."]);

  return (
    <div>
      <input value={search}
         onChange={(e) => setSearch(e.target.value)} />
      <ShowTodoList list={todoList} search={search} />
    </div>
  );
}
```

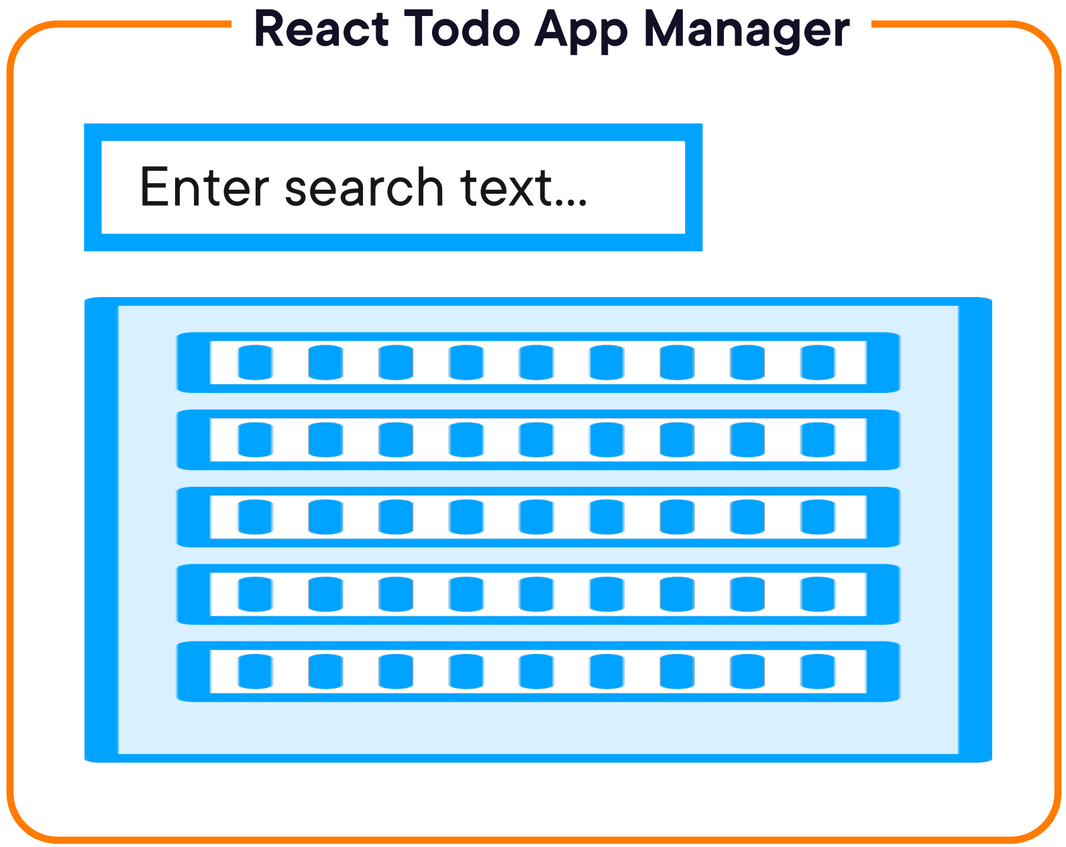
Typical React App (Todo Manager)

```
function App() {
  const [search, setSearch] = useState("");
  const deferredSearch = useDeferredValue(search);

  const [todoList, setTodoList] = useState([
    "clean dog","eat lunch","wash clothes","..."]);

  return (
    <div>
      <input value={search}
         onChange={(e) => setSearch(e.target.value)} />
      <ShowTodoList list={todoList} search={deferredSearch} />
    </div>
  );
}
```

Prioritizing State Updates in Components with useTranstion

The React API includes a library call useTranstition to explicitly lower the priority of state updates

Typical React App (Todo Manager)



```
function App() {
  const [search, setSearch] = useState("");
  const [todoList, setTodoList] = useState([
          "clean dog", "eat lunch", "wash clothes", "..."]);

  return (
    <div>
      <input value={search} onChange={(e) => {
        setSearch(e.target.value)
      }} />
      <ShowTodoList todoList={todoList} deferredSearch={search} />
    </div>
  );
}
```

```
function App() {
  const [search, setSearch] = useState("");
  const [isPending, startTransition] = useTransition();
  const [todoList, setTodoList] = useState([
          "clean dog", "eat lunch", "wash clothes", "..."]);

  return (
    <div>
      <input value={search} onChange={(e) => {
        setSearch(e.target.value)
      }} />
      <ShowTodoList todoList={todoList} deferredSearch={search} />
    </div>
  );
}
```

```
function App() {
  const [search, setSearch] = useState("");
  const [searchHighPriority, setSearchHighPriority] = useState("");
  const [isPending, startTransition] = useTransition();
  const [todoList, setTodoList] = useState([
          "clean dog", "eat lunch", "wash clothes", "..."]);

  return (
    <div>
      <input value={search} onChange={(e) => {
        setSearch(e.target.value)
      }} />
      <ShowTodoList todoList={todoList} deferredSearch={search} />
    </div>
  );
}
```

```
function App() {
  const [search, setSearch] = useState("");
  const [searchHighPriority, setSearchHighPriority] = useState("");
  const [isPending, startTransition] = useTransition();
  const [todoList, setTodoList] = useState([
          "clean dog", "eat lunch", "wash clothes", "..."]);

  return (
    <div>
      <input value={search} onChange={(e) => {
         startTransition(() => setSearch(e.target.value))}
      } />
      <ShowTodoList todoList={todoList} deferredSearch={search} />
    </div>
  );
}
```

```
function App() {
  const [search, setSearch] = useState("");
  const [searchHighPriority, setSearchHighPriority] = useState("");
  const [isPending, startTransition] = useTransition();
  const [todoList, setTodoList] = useState([
          "clean dog", "eat lunch", "wash clothes", "..."]);

  return (
    <div>
      <input value={search} onChange={(e) => {
         setSearchHighPriority(e.target.value);
         startTransition(() => setSearch(e.target.value))}
      } />
      <ShowTodoList todoList={todoList} deferredSearch={search} />
    </div>
  );
}
```

```
function App() {
  const [search, setSearch] = useState("");
  const [searchHighPriority, setSearchHighPriority] = useState("");
  const [isPending, startTransition] = useTransition();
  const [todoList, setTodoList] = useState([
          "clean dog", "eat lunch", "wash clothes", "..."]);

  return (
    <div>
      <input value={searchHighPriority} onChange={(e) => {
         setSearchHighPriority(e.target.value);
         startTransition(() => setSearch(e.target.value))}
      } />
      <ShowTodoList todoList={todoList} deferredSearch={search} />
    </div>
  );
}
```

Implement useDeferredValue in the Todo App

Implement useTransition in the Todo App

useTransition in Action Frame by Frame

Takeaways

Avoid temptations to optimize everything

Easy to find over-rendering with React Devtools Flamegraph

Avoid prioritizing state updates when not necessary

Do optimize though when your browser does have performance issues

Takeaways

When you have a hammer, everything looks like a nail