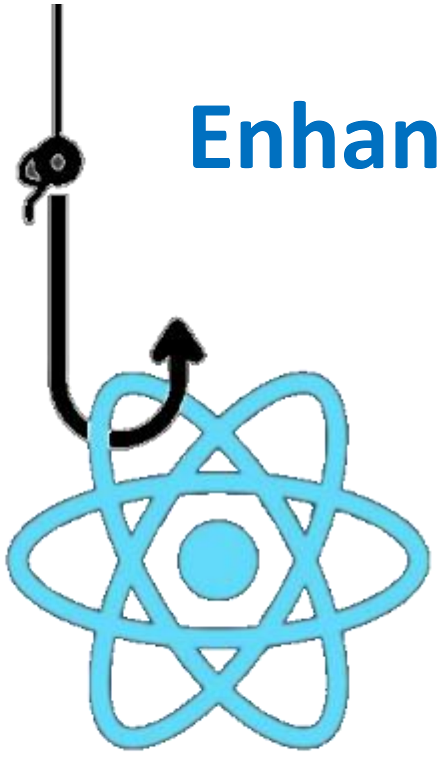
Enhance the Performance of React Apps

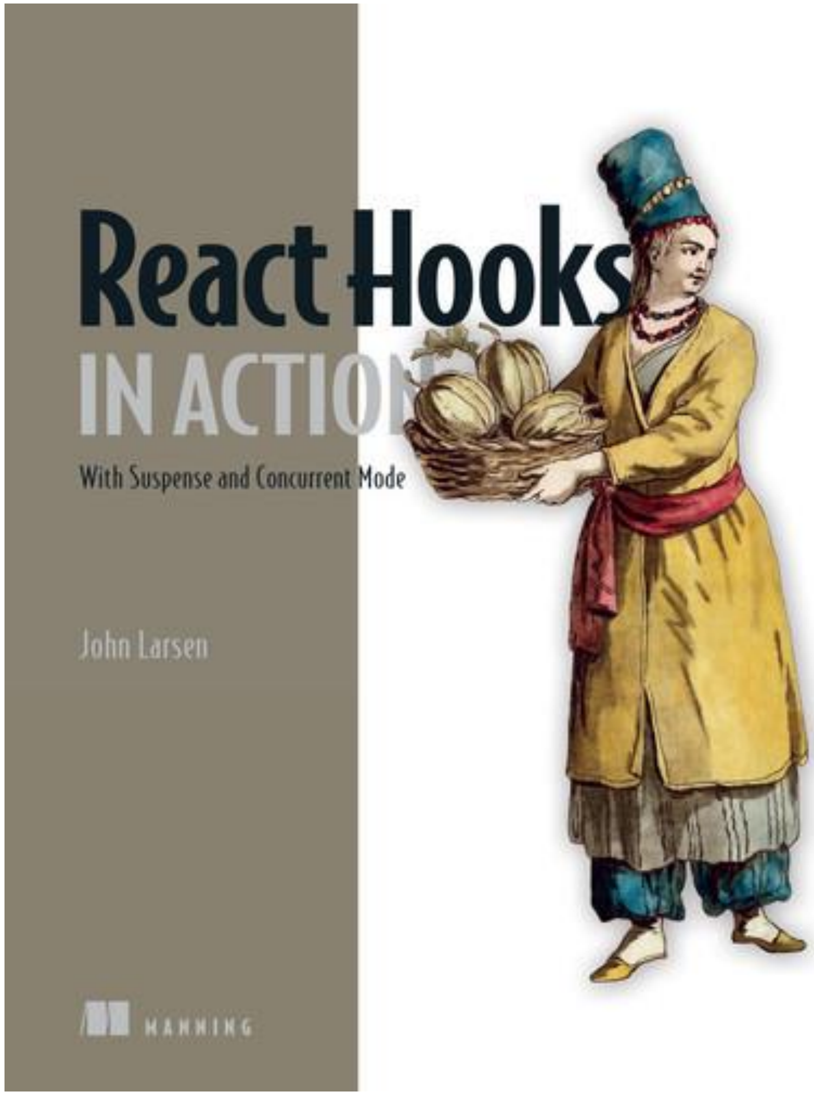


Outline

1. useMemo
2. useCallback
3. useTransition
4. <Suspend> Component

Aim = improve the user experience of our apps, making them feel more responsive

Slides are based on



React Hooks IN ACTION

With Suspense and Concurrent Mode

John Larsen

MANNING

https://learning.oreilly.com/library/view/react-hooks-in/9781617297632/

useMemo

- Avoid unnecessarily rerunning of expensive computations by wrapping them in the useMemo hook
    - Pass useMemo the expensive function you want to memorize and control re-execution with a dependency array

```
const value = useMemo(
  () => expensiveFn(dep1, dep2), [dep1, dep2]
);
```

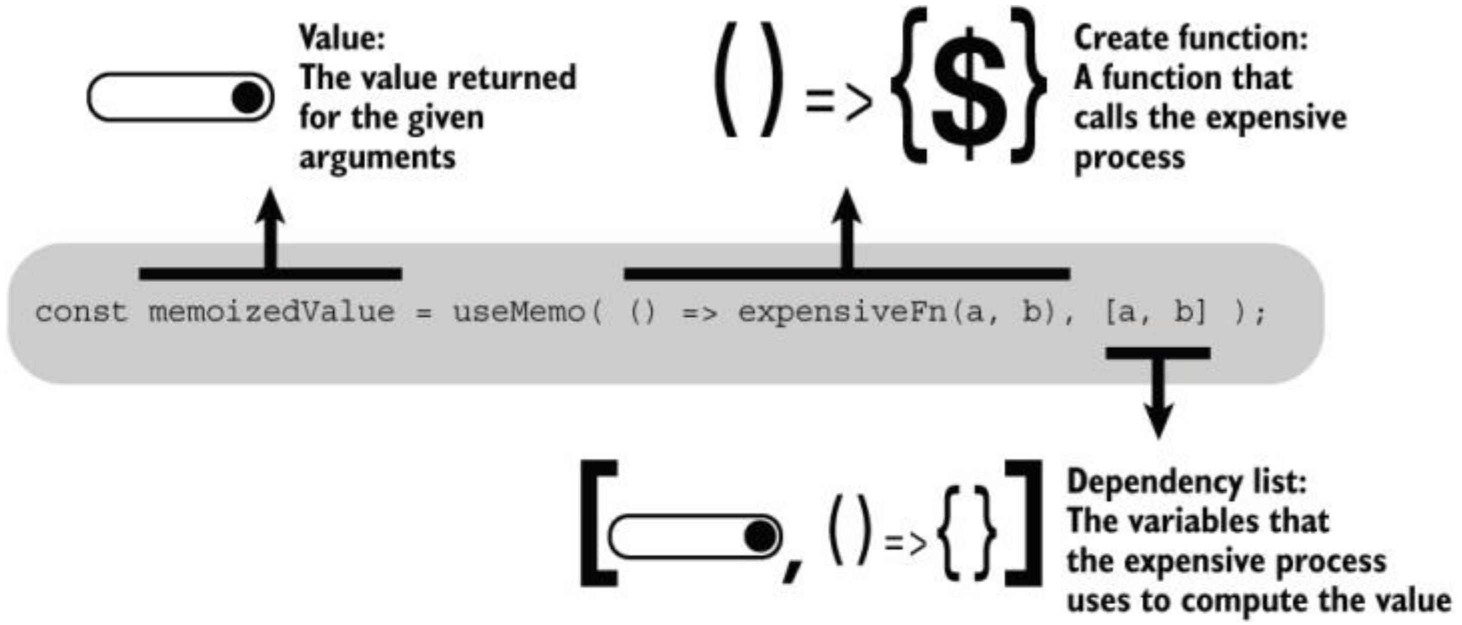
- Use the **memo** hook to avoid unnecessary re-rending of components
    - Help us avoid unnecessary and wasteful work to protect the user from sluggish UI updates

Memoizing

- By calling the function inside the useMemo hook, we ask React to
    - store a value computed by the function for a given set of arguments
    - If we call the function inside useMemo again, using the same arguments as the previous call, it should return the stored value
    - If we pass different arguments, it will use the function to compute a new value and update its store before returning the new value
- The process of storing a result for a given set of arguments is called memoizing

Memoizing expensive function calls with useMemo

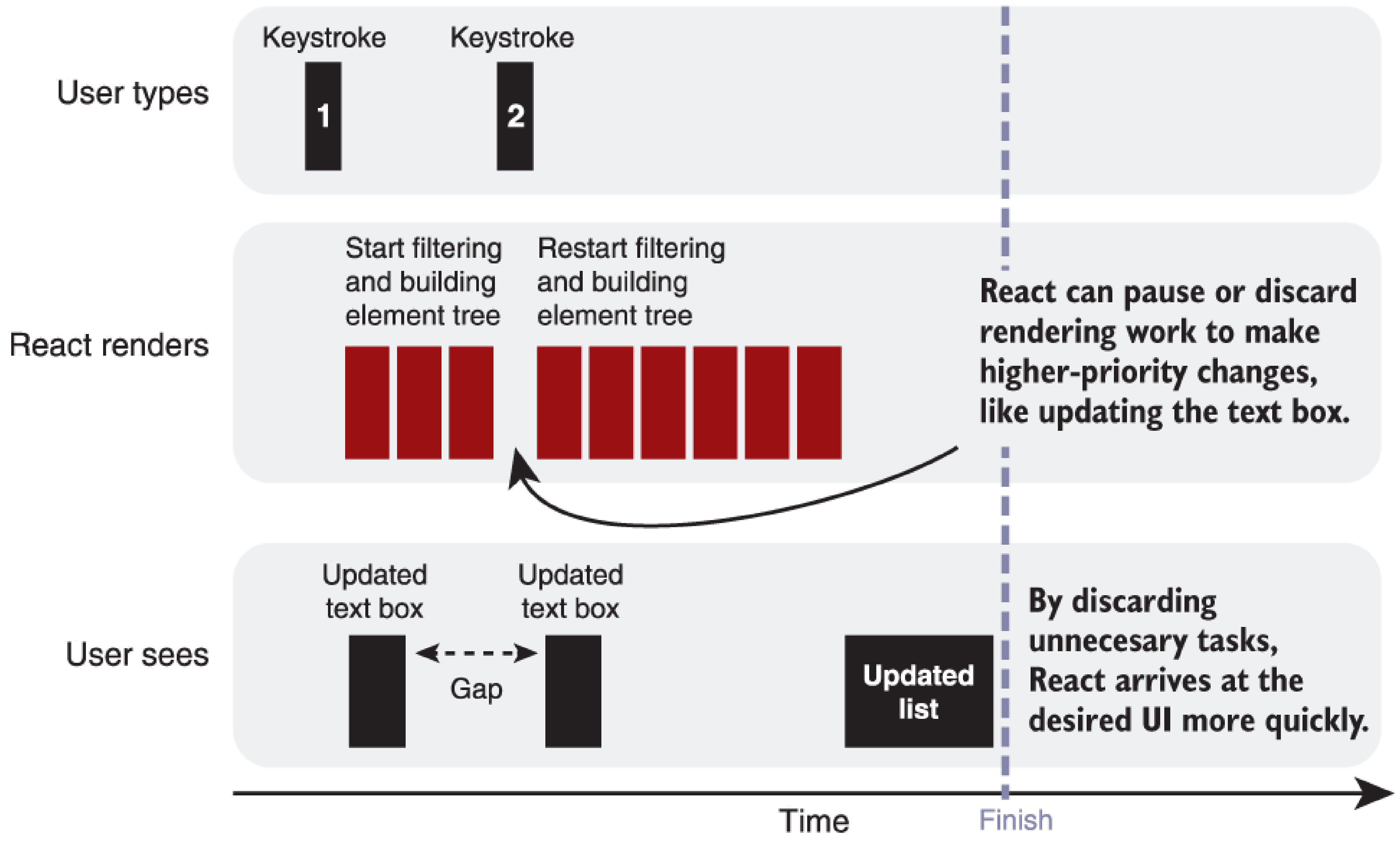
- Wrap expensive functions inside a useMemo hook and control re-execution with a dependency array
  - If the variables in the dependency array don’t change from one call to the next, useMemo returns its stored result for the expensive function


Value:
The value returned for the given arguments
() => {$}
Create function:
A function that calls the expensive process
const memoizedValue = useMemo(() => expensiveFn(a, b), [a, b]);
[, () => {}]
Dependency list:
The variables that the expensive process uses to compute the value

useTransition

- Use useTransition to request React to delay (i.e., **lower the priority)** of updating parts of DOM after state changes to make sure it first responds to user interactions
  - E.g., when filtering a list, React can pause rendering of the filtered list to make sure the text that the user is typing appears in the textbox
- Reconciling and committing changes to one part of the component tree can be paused or abandoned to make sure components with higher priority are updated first
  - Enables React can pause longer-running updates to quickly react to user interactions to deliver smoother UI

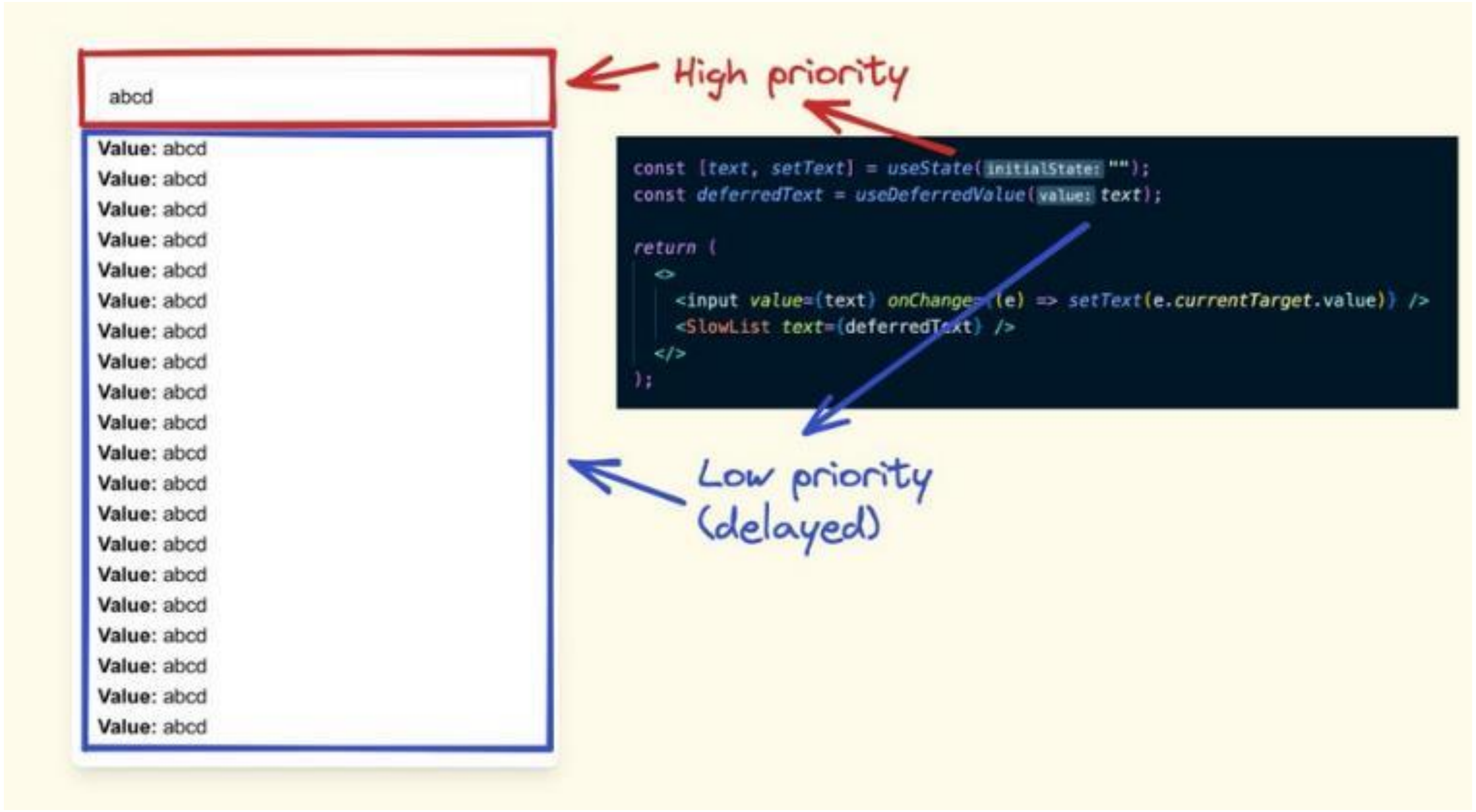
useTransition



User types
Keystroke
1
Keystroke
2
React renders
Start filtering and building element tree
Restart filtering and building element tree
React can pause or discard rendering work to make higher-priority changes, like updating the text box.
User sees
Updated text box
Gap
Updated text box
Updated list
By discarding unnecesary tasks, React arrives at the desired UI more quickly.
Time
Finish

useDeferredValue

- useDeferredValue lets you split one piece of state into two: a **high priority version**, and a low priority one
    - In this example, the input updates immediately, and the SlowList renders whenever it can.



```
abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
Value: abcd
```

```
const [text, setText] = useState(initialState: "");
const deferredText = useDeferredValue(value: text);

return (
  <>
    <input value={text} onChange={(e) => setText(e.currentTarget.value)} />
    <SlowList text={deferredText} />
  </>
);
```

useDeferredValue

- useTransition() wraps the state updating code, whereas useDeferredValue() wraps a value affected by the state change
- If you have access to the **state updating code** and have some state updates that should be treated with a lower priority, it makes sense to use useTransition(). Use useDeferredValue() if you don’t have that access
- UseTransition() and useDeferredValue() should not be used to wrap up all of your state updates or values.
    - Only use if a component **can’t be optimized any other way**. Other performance enhancements, such as lazy loading, pagination, and performing work in worker threads or on the back end, should always be considered.

Suspend Component

- Suspense component allows rendering **fallback content** to indicate that a component is waiting for something, like loading data needed to build the UI

```
<Suspense fallback={<MySpinner />}>
   <MyFirstComponent />
   <MySecondComponent />
</Suspense>
```

- Suspense allows showing fallbacks either for individual components or groups of components

Suspend Component

- Suspend component wraps a component, which is loading the data from some data source, and it will show a **fallback** until the data fetching is complete

Summary

- **useMemo:** avoid rerunning expensive computations
- **useTransition**: React can pause longer-running updates to quickly react to user interactions
- **Suspense** component allows rendering **fallback content** to indicate that a component is waiting for something, like loading data

Resources

- Thinking in React

https://reactjs.org/docs/thinking-in-react.html

- React Router

https://reactrouter.com/

- Useful list of resources

https://github.com/enaqx/awesome-react