React State

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi International University.

Content

- State
- State as a snapshot
- Event handler
- Passing method references between components
- Lifting state up
- Updater function

State

State plays 2 role:

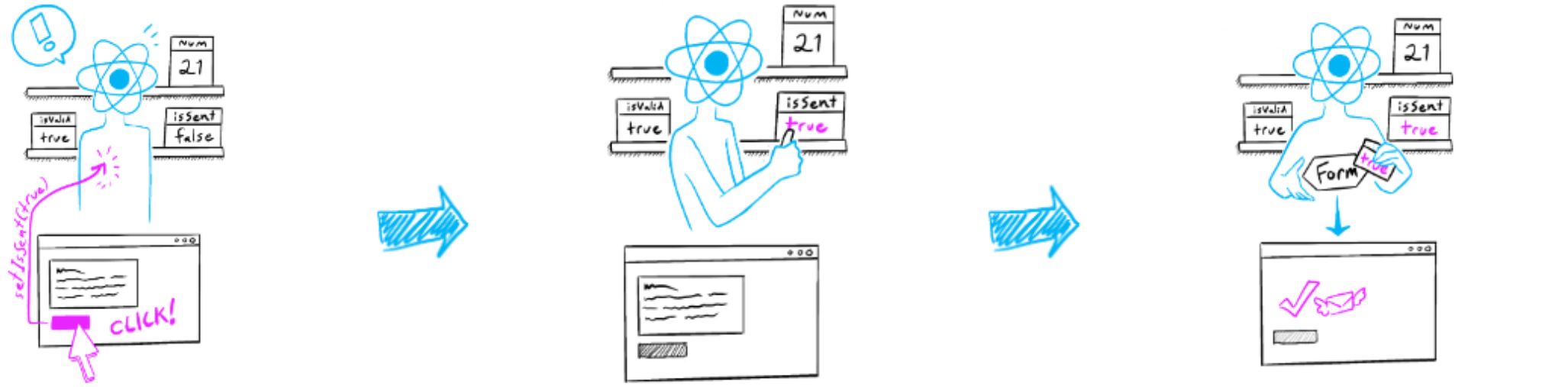
1. Storing data in the component such as user input filled in the form, flags such as data is sent to the server successfully or not.
2. When you update state correctly with setter function, it triggers UI update.

State is used in a component and managed by React.

State is data inside component that triggers UI update

“Rendering” means that React is calling your component, which is a function. The JSX you return from that function is like a snapshot of the UI in time.

Illustrated by Rachel Lee Nabors



You tell React to update the state

React updates the state value

React passes a snapshot of the state value into the component

useState hook

- Returns an array with 2 elements.
  - The state as the first element via which you can access the state value.
  - Sets the state and triggers the UI update.
- Pass the default value for the state as a param to the useState hook.

```
const [count, setCount] = useState(0);
```

Learn more: using the state hook

When you call setState, under the hood

1. When click on the button, your component sends a signal with setState function to React to update the state.
2. React updates the state.
3. Your component returns the JSX with the update state which is a snapshot of the UI at a time.
4. React compares the previous screen with the new snapshot and renders only required portion in the UI. This is also known as new and old virtual DOMs. Lets discuss them tomorrow.

Snapshots

SNAPSHOT 1 (HTML) – When page loads for the first time.

```
<div class="App">
        <p>Counter is: 1</p>
        <button>Increment by one</button>
</div>
```

SNAPSHOT 2 (HTML) – After clicking on the button.

```
<div class="App">
        <p>Counter is: 2</p>
        <button>Increment by one</button>
</div>
```

State as a snapshot

Unlike regular JavaScript variables, React state behaves more like a snapshot. Setting it does not change the state variable you already have, but instead triggers a re-render. This can be surprising at first!

```
console.log(count); // 0
setCount(count + 1); // Request a re-render with 1
console.log(count); // Still 0!
```

This could help from bugs. For more: https://react.dev/learn/adding-interactivity#state-as-a-snapshot

Copy state before setState

When you have an object or array state, you must copy the state before making changes.

Because they can have N number of elements. For performance reason, React don’t scan every single element in an array of object.

Instead React does a shallow comparison. Meaning it will only check the memory addresses of the state and will not conduct comparisons deeper into the properties.

For more: Shallow Comparison vs Deep Comparison in Javascript

React batches state updates

For performance optimization, React updates states only once for each render.

This is uncommon to update a single state multiple times in one event. But if you want to that, you can use an **updater function**.

setNumber(number + 1) to setNumber(n => n + 1).

For more: https://react.dev/learn/queueing-a-series-of-state-updates

Lifting state up

Sometimes, you want the state of two components to always change together. To do it, remove state from both of them, move it to their closest common parent, and then pass it down to them via props. This is known as lifting state up, and it’s one of the most common things you will do writing React code.

As we progress, you will write code that requires lifting state up.

For more: https://react.dev/learn/sharing-state-between-components#lifting-state-up-by-example

Summary

- If state is an array or an object, always copy before updating.
- You can pass an event handler as a prop.
- Try updater function in the setState if you are updating the state multiple times in one event handler.
- Designing state is similar to normalizing the database. Group related state, avoid redundancy and duplication. Keep it simple and easy to manage. For more: https://react.dev/learn/managing-state
- With setState, the component function will be called, also all its subsequent child components will be called regardless of whether their props have changed or not.