List & Styling

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Conditionals
- Lists
- Immutable state and how React updates DOM
- Style

Conditional Rendering

Your components will often need to display different things depending on different conditions. In React, you can conditionally render JSX using JavaScript syntax like if statements, &&, and ? : operators.

```
function Item({ name, isPacked }) {
  return (
    <li className="item">
      {name} {isPacked && '✓'}
    </li>
  );
}
```

Rendering Lists

You will often want to display multiple similar components from a collection of data. You can use the JavaScript array methods to manipulate an array of data. You can use filter() and map() with React to filter and **transform your array of data into an array of components**.

Rendering Lists

Data to render as a list:

```
const students = [
        { id: 1234, name: "s1", age: 12 },
        { id: 9994, name: "s2", age: 11 },
        ...
]
```

To render on UI, map the JSON to a component in map:

```
students.map(student => <Student name={student.name} age={student.age} />)
```

Key when rendering a list

Keys tell React which array item each component corresponds to, so that it can match them up later. This becomes important if your array items can move (e.g. due to sorting), get inserted, or get deleted.

Rules of key:

1. Key must be unique
2. Key cannot be changed

Keys aren't just for lists! You can use keys to make React distinguish between any components. To reset state, you can give components different keys. For more: Resetting state with a key

Quick Recap

- Objects and arrays are reference types.
- When we mutate an array or object like `const students = this.state.students;` Both are pointing to the same reference.

- When we change something on students, we also change their original state.

- To put it another way, we change the original data. React doesn't know if state has changed when we modify it because it still points to the same memory address. Thus, the page doesn't display any fresh data.

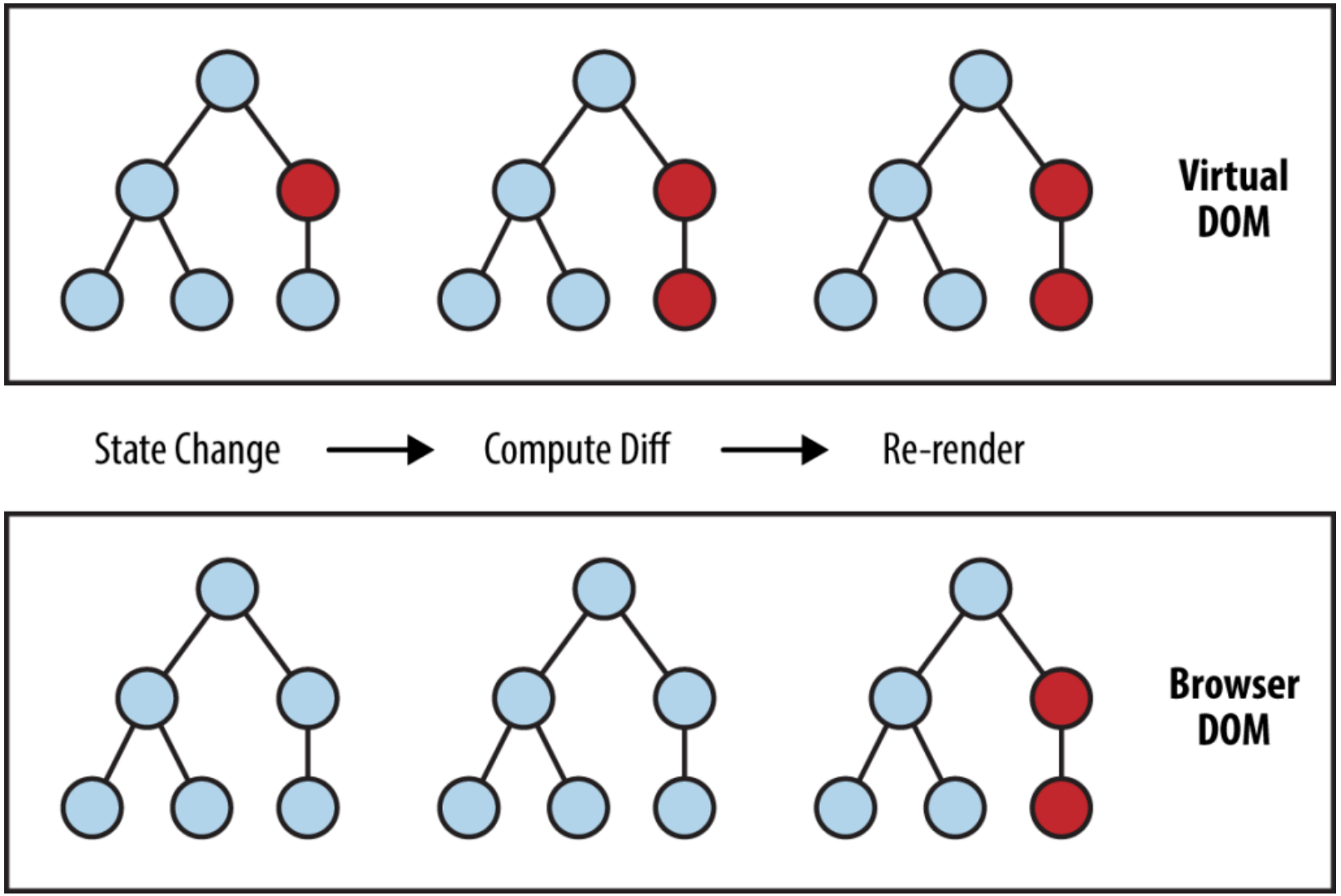
- Never mutate state! What should we do? Copy the array or object. Or use array functions that returns a new array without mutating the original array.

How React Updates DOM

- In JavaScript, virtual DOM is DOM representation. Because nothing is drawn onscreen, manipulating the virtual DOM is much faster.

- The old virtual DOM and the new virtual DOM are both kept in memory by React.
- It compares the differences between the old and new virtual DOM trees (new snapshot).

- If there is a difference, it only renders the difference to real DOM. The actual DOM is the web page that you see in your browser.
- Again, not all of the DOM is rendered. Only the differences are rendered.
- For more details: React Fiber Architecture



source: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch02.html

Immutables

Before the update, React compares the old virtual DOM to the new virtual DOM to determine what changed. This is the process of reconciliation.

If the state consists of immutable objects, you can use a simple equality operator to see if they have changed (AKA shallow comparison). Immutability, in this sense, removes complexity.

Because knowing what changed can be difficult at times.
Consider the following: myPackage.sender.address.country.id = 1;

Because an immutable value or object cannot be changed, each update generates a new value while leaving the old one unchanged. Immutability is embraced by array methods.

Add an element to immutable Array

Use ‘concat() and filter()’ which return new array. Embrace immutability. They don’t change the orignial array.

```
const array1= [1,2,3];
const array2 = array1.concat(4);
```

```
const array1= [1,2,3];
const array2 = array1.filter(item=> item !=2);
```

Manipulating State Immutably

Splice, unlike map and filter,

does not return a new array. As a result, we must copy it before making changes.

As a result, React is aware that

the state has changed.

```
students =[
      { name: 'Alice', age: 20 },
      { name: 'Bob', age: 19 },
      { name: 'Jimmy', age: 21 }
    ]



 deleteStudent = (index) => {
    const students = [...this.state.students];
    students.splice(index, 1);
    this.setState({ students });
 }
```

React Styling

- Inline style
- External CSS
- Styled-components
- Common properties
- Layout

Inline style

- All properties are in **camelCase** like backgroundColor

```
<p
style={{
backgroundColor: "green",
color: "white",
textAlign: "center"
}}
>
Hello World
</p>
```

External CSS

- All properties are in **kebab-case** like background-color

App.css

p {background-color: green; color: white; text-align: center}

App.js

import "./App.css";

…

* *

Hello World

* *

External CSS – Class Name

- All properties are in kebab-case like background-color

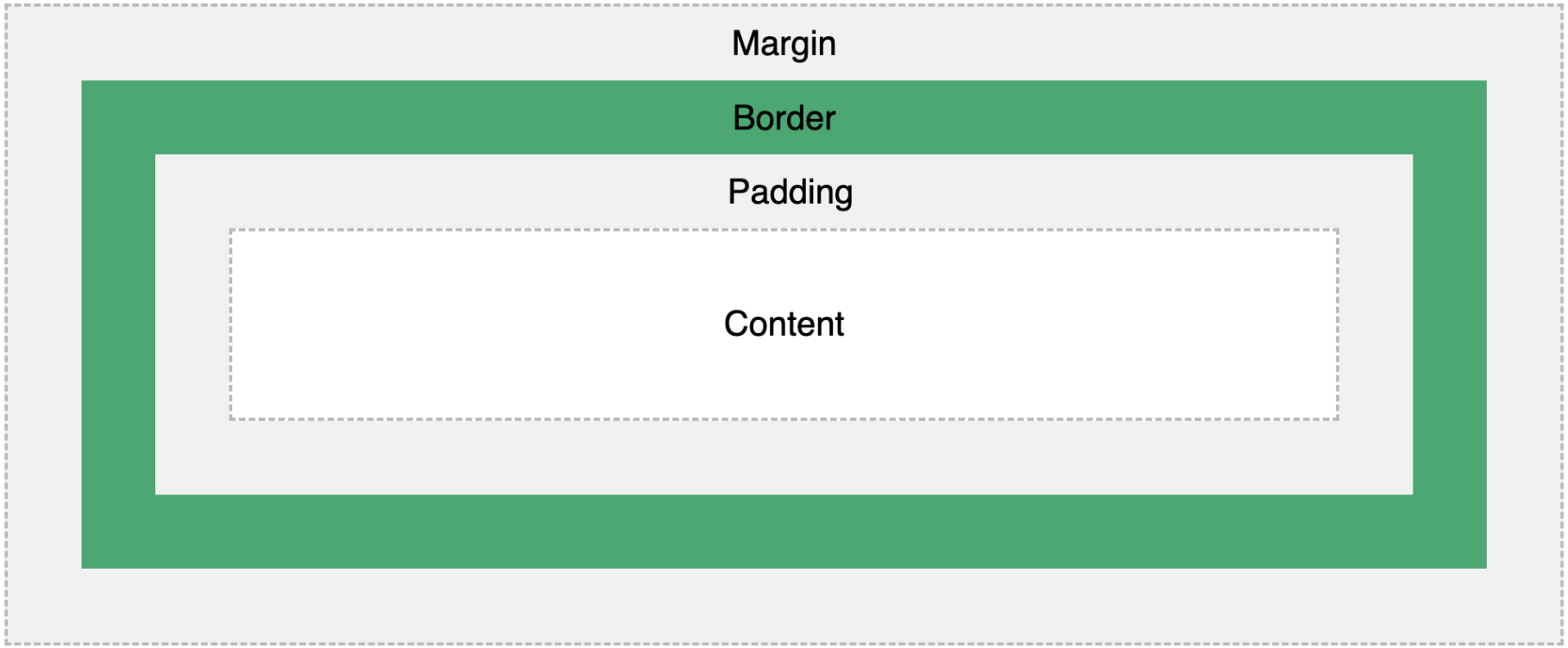
App.css

```
.paragraph{background-color: green; color: white; text-align: center}
```

App.js

```
import "./App.css";
...
<p className="paragraph">
Hello World
</p>
```

Box model



- Refer: https://www.w3schools.com/css/css_boxmodel.asp

Common properties

- **color:** Sets the text color of an element's content.
- **background-color:** Sets the background color of an element.
- **font-size:** Specifies the size of the font used for text. See more: font-family, font-style...
- **margin:** Controls the spacing around an element, creating space outside of the element's border.
- **padding:** Controls the spacing inside an element, creating space between the content and the element's border.
- **border:** Defines the style, width, and color of an element's border.
- **display:** Determines how an element is displayed (e.g., block, inline, flex, etc.).
- **width:** Sets the width of an element.
- **height:** Sets the height of an element.
- **text-align:** Aligns the text content of an element (e.g., left, right, center).

Layout

- Flexbox: https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- Grid: https://css-tricks.com/snippets/css/complete-guide-grid/
- **Always** use Flexbox or Grid to layout your web page
- **Never** use a table or a list for positioning components.

Third parties CSS libraries/framework

- React Bootstrap: https://react-bootstrap.github.io/
- Material UI: https://mui.com/material-ui/
- Tailwind: https://tailwindcss.com/

You only need to use the basic CSS for this course. Please do not use it in this course except your final project.

Summary

- Conditionals
- Lists
- Immutable state and how React updates DOM
- Style