Conditionals & Lists

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

Rendering Content Conditionally

```
state = {
   students: [
     { name: 'Alice', age: 20 },
     { name: 'Bob', age: 19 }
   ],
   showStudents: true
 };

 showHideStudents = () => {
   const showStudents =
this.state.showStudents;
   this.setState({ showStudents:
!showStudents });
 }
```

```
render() {
   return (
     <div className="App">
       <button onClick={this.showHideStudents}>Show / Hide
Students</button>
       {this.state.showStudents ?
         <div>
           <Student name={this.state.students[0].name}
             age={this.state.students[0].age}>
           </Student>
           <Student name={this.state.students[1].name}
             age={this.state.students[1].age}>
           </Student>
         </div> : null
         }
     </div>
   );
 }
```

Better Way

```
render() {
   let students = null;

   if (this.state.showStudents) {
     students = (
       <div>
         <Student
name={this.state.students[0].name}

age={this.state.students[0].age}>
         </Student>
         <Student
name={this.state.students[1].name}

age={this.state.students[1].age}>
         </Student>
       </div>
     )
   }
```

```
    return (
      <div className="App">
        <button onClick={this.showHideStudents}>Show /
Hide Students</button>
        {students}
      </div>
    );

  }
```

Outputting Lists

```
render() {
   return (
     <div className="App">
       {
         this.state.students.map(student => {
           return (
             <Student name={student.name}
               age={student.age}>
             </Student>

...
```

Outputting List

Warning: Each child in a list should have a unique "key" prop. The key prop boosts the performance by not rendering all children over and over again if there is a change. Instead, it only renders the new children.

```
render() {
   return (
     <div className="App">
       {
         this.state.students.map(item => {
           return (
             <Student
               key={item.id}
               name={item.name}
               age={item.age}>
             </Student>
...
```

Updating State

```
state = {
  students: [
    { name: 'Alice', age: 20 },
    { name: 'Bob', age: 19 },
    { name: 'Jimmy', age: 21 }
  ]
};
// incorrect example, explain later
deleteStudent = (index) => {
  const students = this.state.students;
  students.splice(index, 1);
  this.setState({ students });
}
```

```
render() {
   return (
     <div className="App">
       {
         this.state.students.map((item, index) => {
           return (
             <Student
               key={index}
               name={item.name}
               age={item.age}
               myClickHandler={() =>
this.deleteStudent(index)}
             >
             </Student>
           )
         })
       }
     </div>
   );
 }
```

Quick Recap

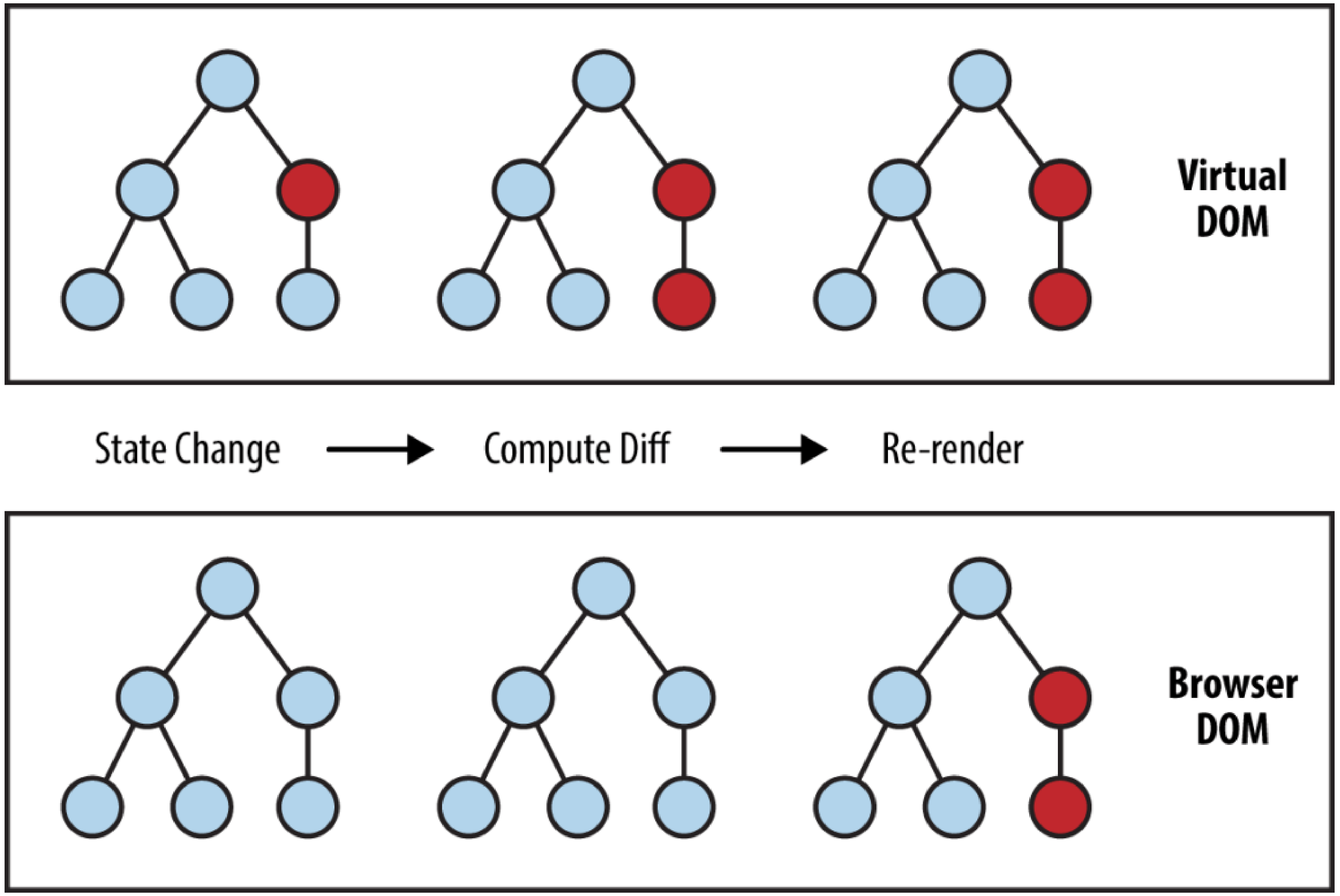
- Objects and arrays are reference types.
- `const students = this.state.students;` Both are pointing the same reference now.
- When we change something on students, original state also gets changed.
- In other words, we mutate the original data.

What should we do?

- Copy the array.
- Then change it.
- Remember I: slice() without arguments copies the array
- Remember II : spread operator can be used.

How React Updates DOM

- Virtual DOM is DOM representation in JavaScript. Manipulating the virtual DOM is much faster, because nothing gets drawn onscreen.
- React keeps **2 virtual DOM**. (the old virtual DOM and the new virtual DOM)
- It **compares** the differences between the old virtual DOM and the new virtual DOM.
- If there is a change between them, then it renders it to real DOM
- It does not render all the DOM. **It only renders the differences**.



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

Immutables

React compares the old virtual DOM with the new virtual DOM before the update in order to know what changed. This is the **reconciliation process**.

If the state is immutable objects, you can check if they changed with a simple equality operator. From this perspective, immutability removes complexity.

Because sometimes, knowing what changed can be very hard. Think about deep fields: myPackage.sender.address.country.id = 1;

An immutable value or object cannot be changed, so every update creates new value, leaving the old one untouched. Array methods embrace immutability.

Add an element to Immutable Array

Use ‘concat()’ which returns new array

```
const array1= [1,2,3];

const array2 = array1.concat(4);
```

Remove from Immutable Array

Use filter()’ which returns new array

```
const array1= [1,2,3];

const array2 = array1.filter(item=> item !=2);
```

Manipulating State Immutably

```
state = {
  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 });
}
```