Week 4 – React Library

Dr Kiki Adhinugraha

Java Script Libraries/Frameworks

- Prototype Framework
- Dojo
- The Yahoo! UI Library
- jQuery
- Angular
- React
  - This is the JavaScript library we will be using in this subject
- Etc.

Why use a library?

- Why invent code when someone has already written it?
- Take inspiration from other coders.
- A well-written library can take away some of the headaches of writing cross-browser JavaScript

Prototype Framework

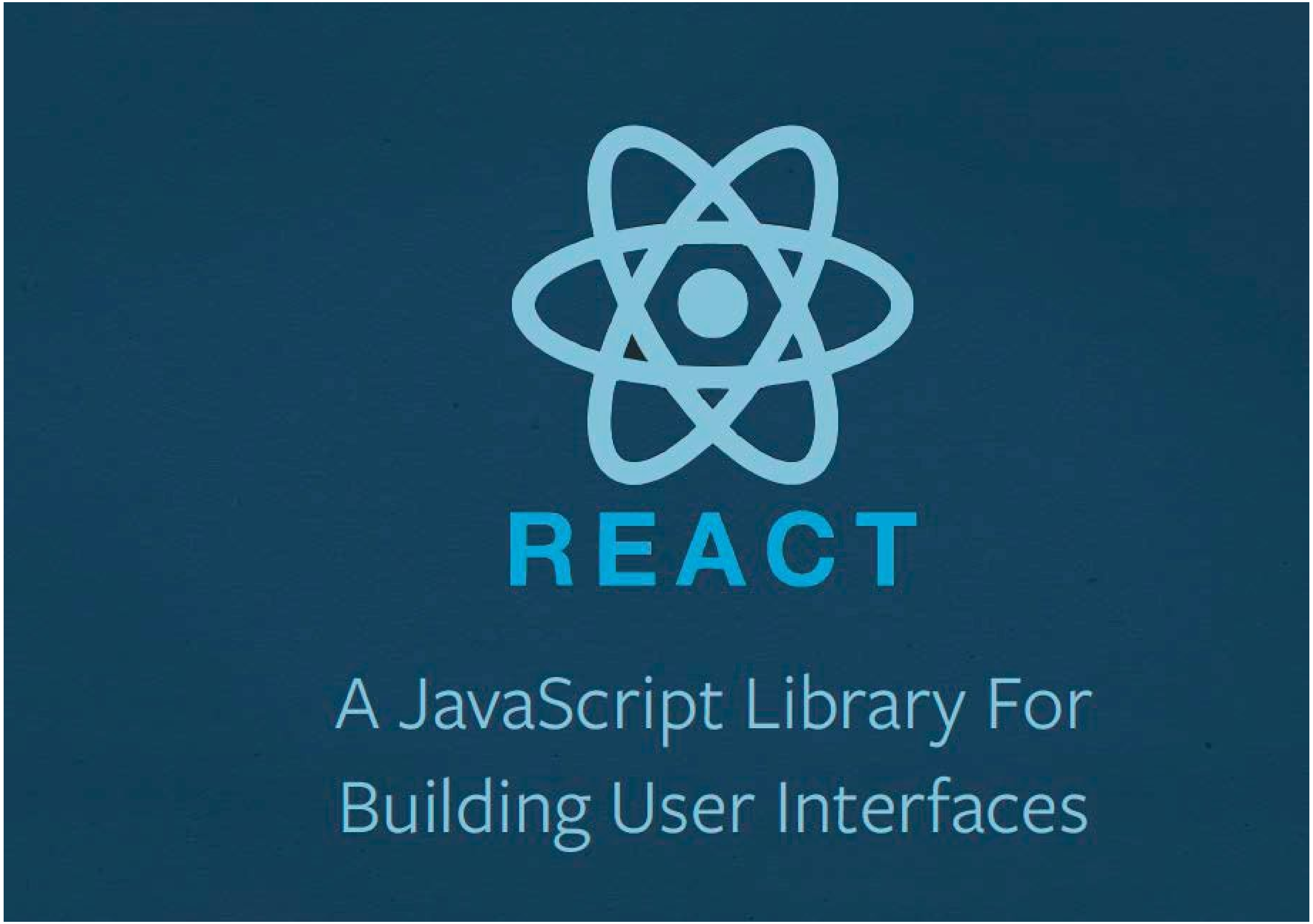
- Prototype extends JavaScript with a number of features.
- These features include
  - Defining classes and inheritance
  - Added custom methods to DOM element nodes
  - Simplifies the most common kinds of Ajax requests
  - Support for JSON encoding and decoding
  - Event delegation

- Due to time constraints we will not be able to go into these features. However, you are welcome to look at the following web site to learn more.
  - http://prototypejs.org/learn/

React

- This is the JavaScript library that we will be using, hence we will go into this in more detail.
- This is a recent library developed by Facebook.
- It has rapidly become very popular.

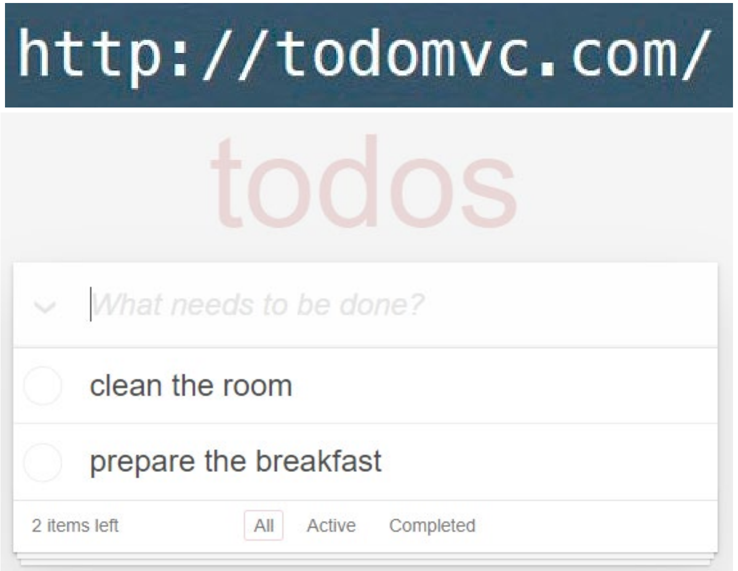
- Many of the examples and slides for React come from the this excellent talk:
  - https://www.youtube.com/watch?v=m2fuO2wl_3c
- The examples have been updated to the latest version of React.



REACT

A JavaScript Library For Building User Interfaces

Example of Interactive Frontend



http://todomvc.com/

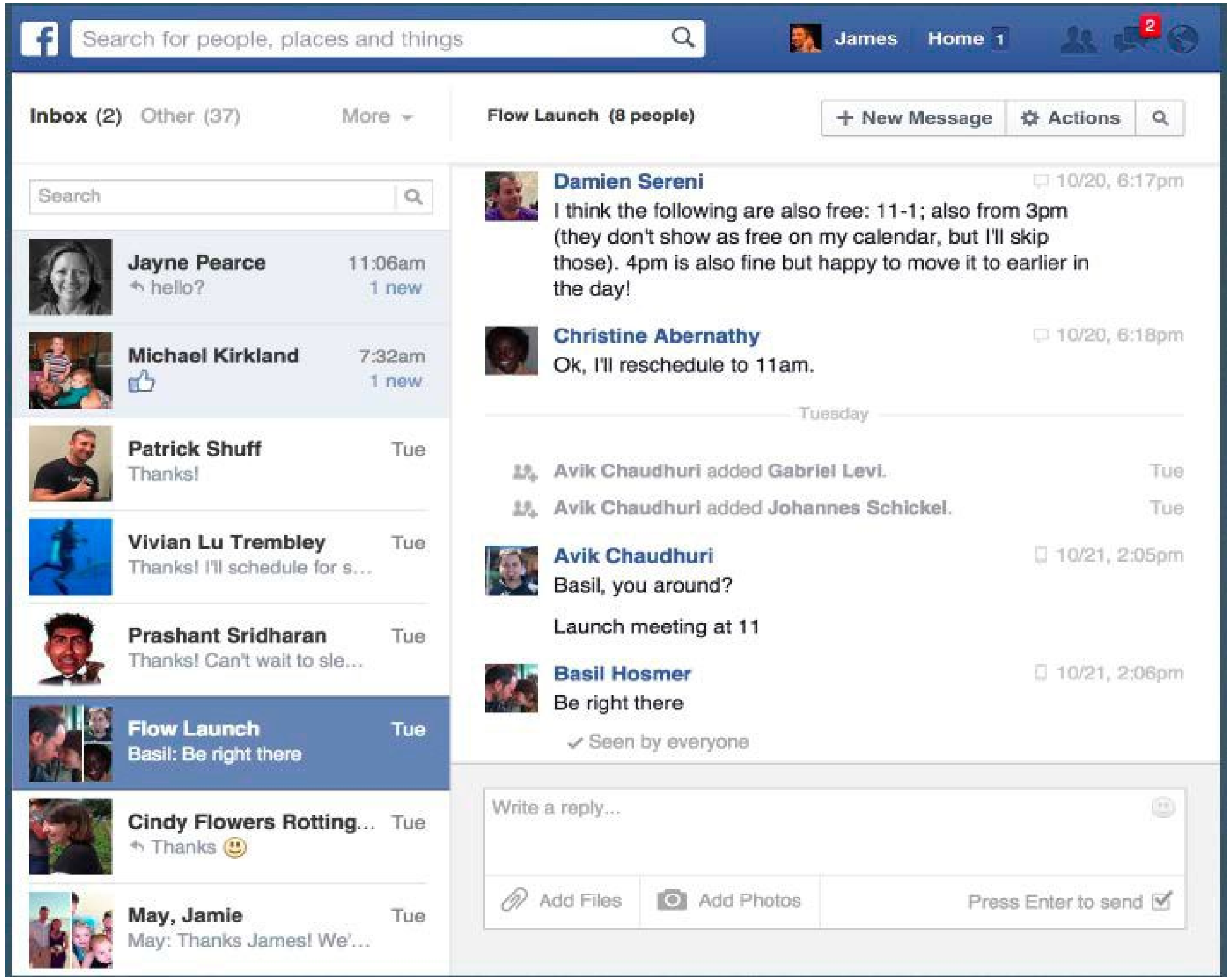
todos

What needs to be done?

clean the room

prepare the breakfast

2 items left



Jayne Pearce 11:06am
hello? 1 new

Michael Kirkland 7:32am
1 new

Patrick Shuff Tue
Thanks!

Vivian Lu Trembley Tue
Thanks! I'll schedule for s...

Prashant Sridharan Tue
Thanks! Can't wait to sle...

Flow Launch Tue
Basil: Be right there

Cindy Flowers Rotting... Tue
Thanks 🙂

May, Jamie Tue
May: Thanks James! We'...

Damien Sereni 10/20, 6:17pm
I think the following are also free: 11-1; also from 3pm (they don't show as free on my calendar, but I'll skip those). 4pm is also fine but happy to move it to earlier in the day!

Christine Abernathy 10/20, 6:18pm
Ok, I'll reschedule to 11am.

Tuesday

Avik Chaudhuri added Gabriel Levi. Tue

Avik Chaudhuri added Johannes Schickel. Tue

Avik Chaudhuri 10/21, 2:05pm
Basil, you around?

Launch meeting at 11

Basil Hosmer 10/21, 2:06pm
Be right there

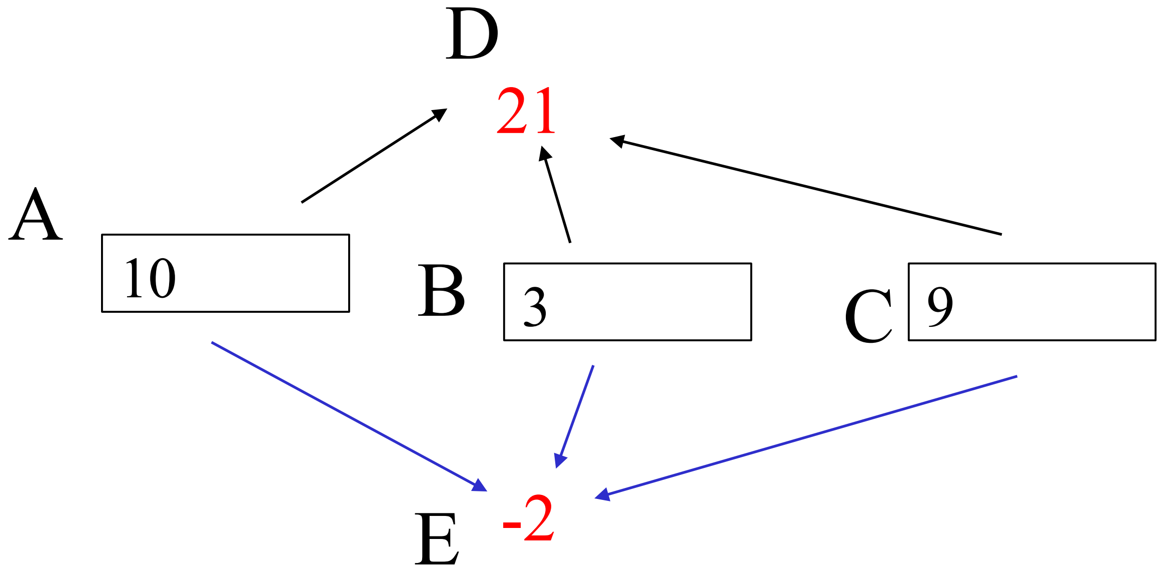
✓ Seen by everyone

React is declarative programming

- Traditional JavaScript libraries/frameworks take the imperative programming approach
  - Describe computation in terms of statements that change a program state.

- React takes a declarative programming approach
  - Express the logic of a computation without describing control flow.

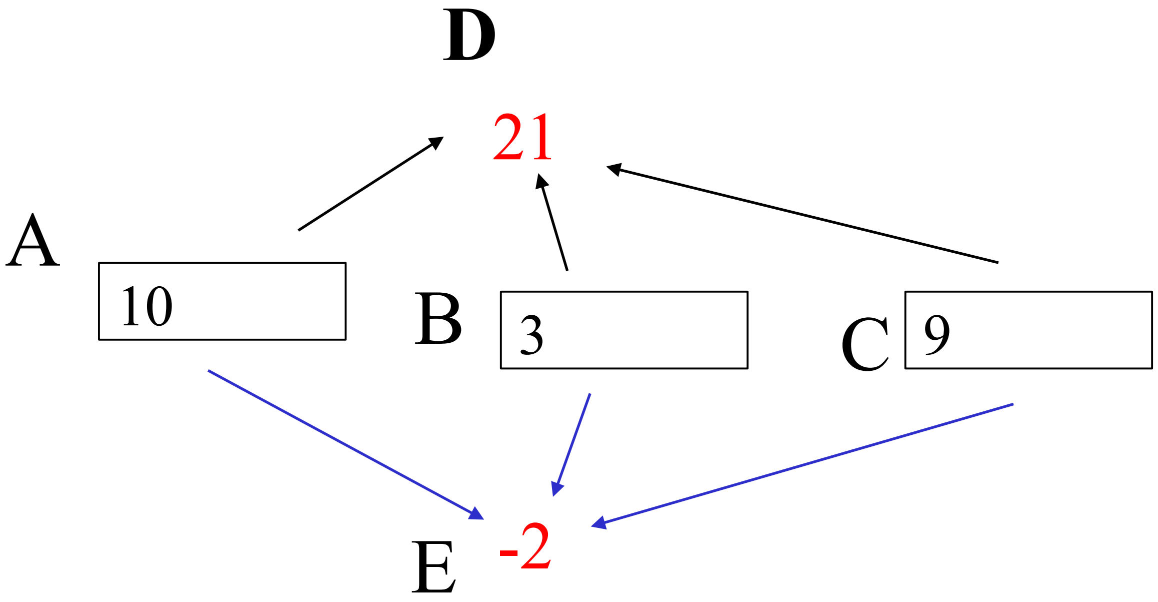
Why is traditional imperative programming approach bad?



- In the above example suppose
  - Element D = A + B + C
  - Element E = A – B – C
- When using imperative programming approach you need to manually specify how each element (D and E) is updated when the user updates textboxes A, B C.
  - In this example the user need to write 6 separate event handlers.
    - Update D due to A change
    - Update D due to B change
    - Update D due to C change

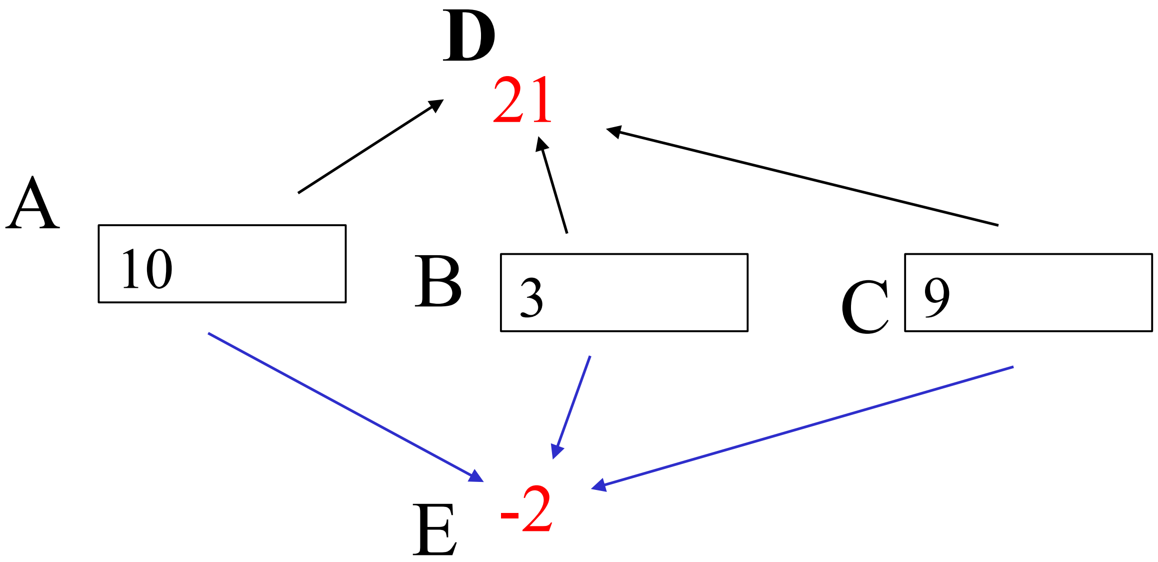
    Update E due to A change
    Update E due to B change
    Update E due to C change
  - Each event handler is a function which specifies exactly how the target element is updated given the updated input elements.

Why is traditional imperative programming approach bad?



- There can be much more complex scenarios than this.
- The programmer can easily forget to write a event handler or make a mistake when writing an event handler.
  - This will leave the web page in an inconsistent state.

React (declarative programming)

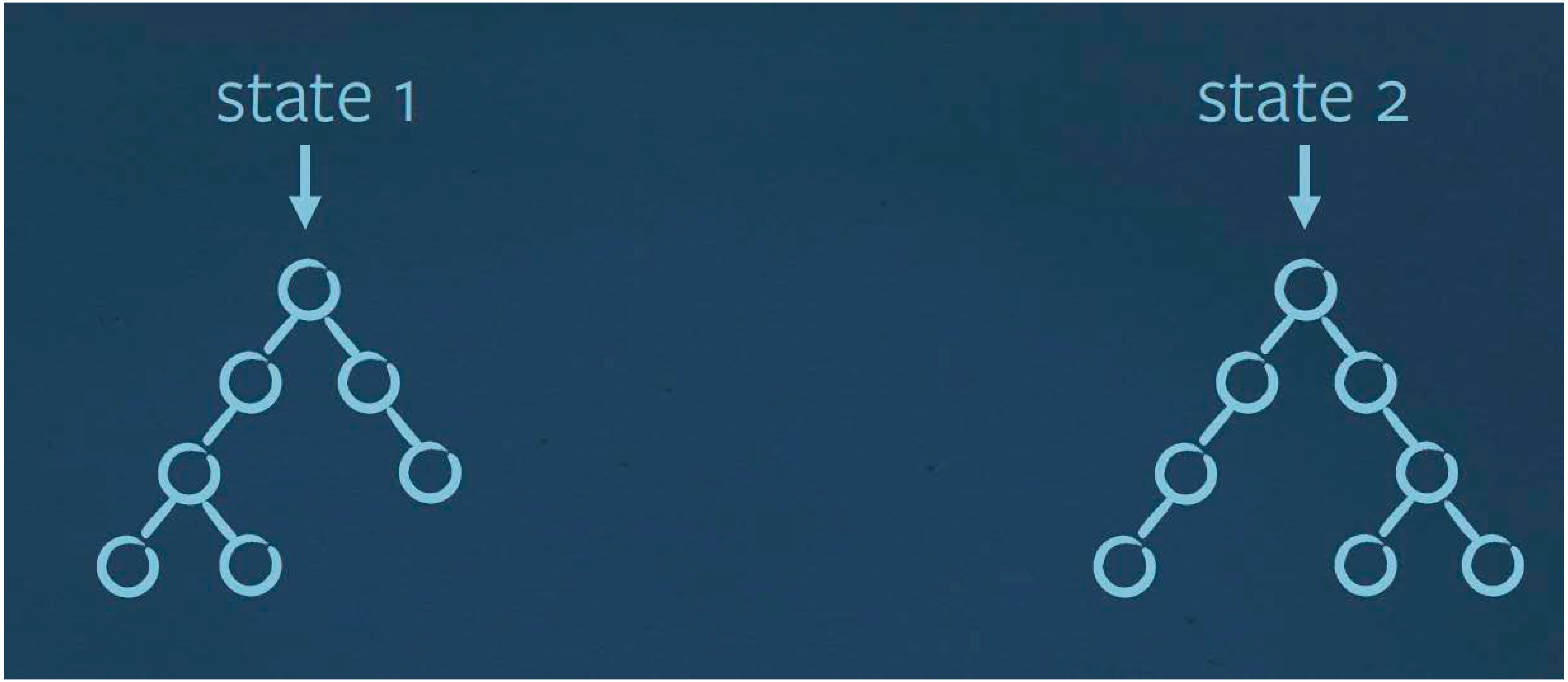


- In react you effectively just specify the formulas for computing elements D, E and F.
  - Element D = A + B + C
  - Element E = A – B – C
- When either A, B and C gets updated. The system will automatically re-render the entire web page! Which means elements D and E will automatically use the updated values in the A, B and C textboxes.

What are you kidding??

- Re-render the entire web page (DOM) on every single small update!
- That must be so so so slow!!!
- Actually not really.

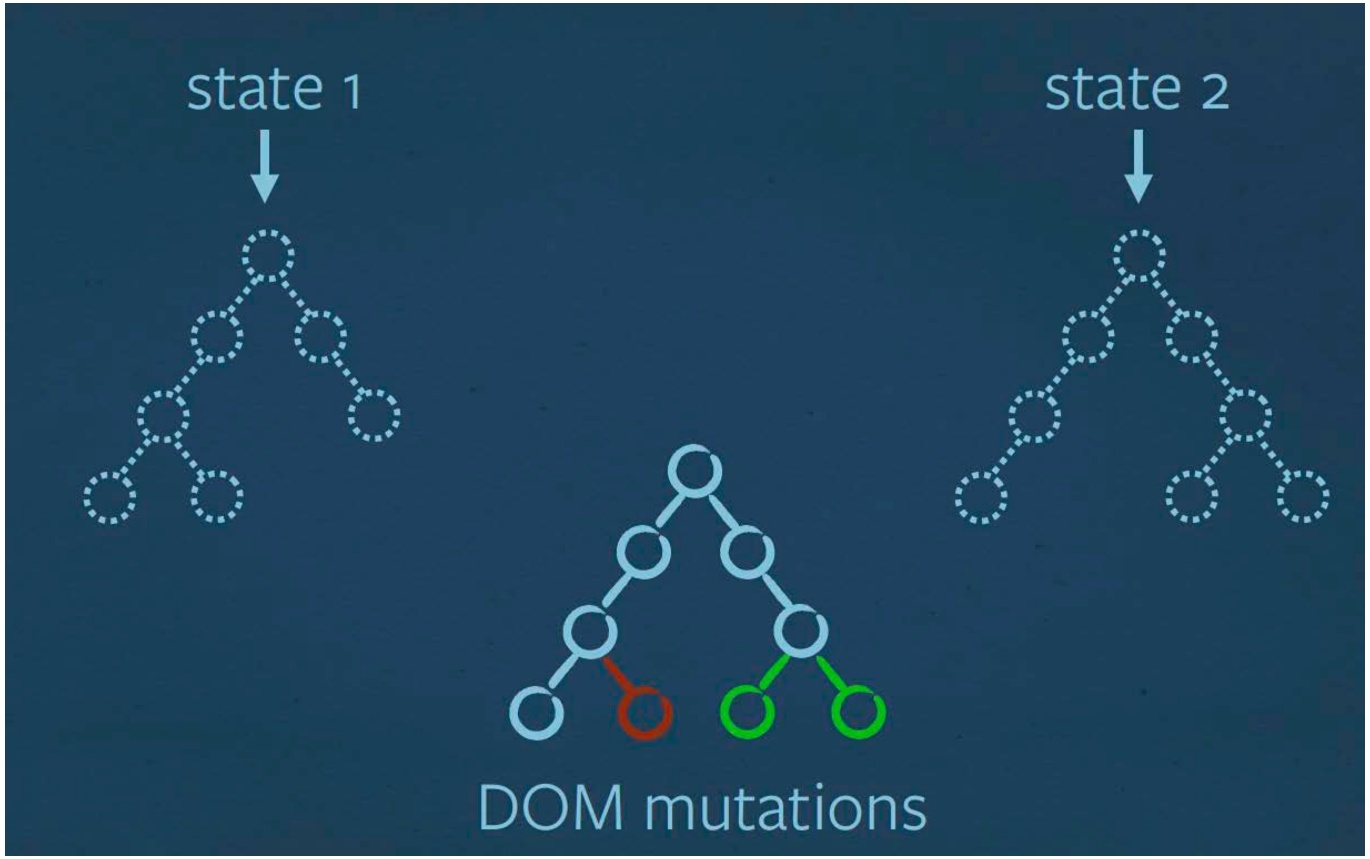
Just update what has changed



- In this example the DOM (tree representing the web page) goes from state 1 to state 2 after the update.

Just update what has changed

- React figures out what has been updated and only renders the updated parts.
  - The way it does this is it constructs an updated virtual DOM
  - (state 2) and computes the difference between the updated virtual DOM (state 2) with the un-updated DOM (state 1).

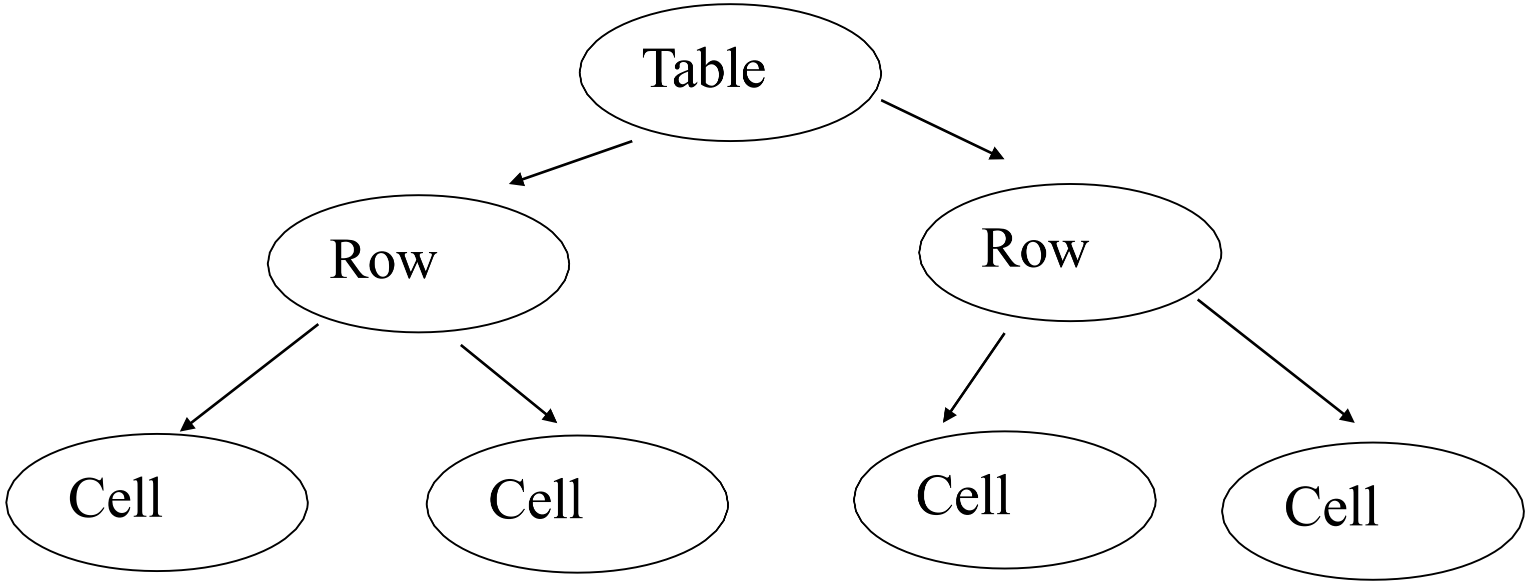


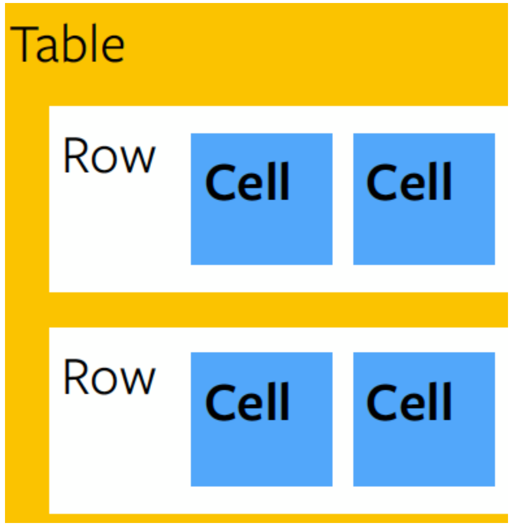
React is very fast

- React is very fast because it just renders the updated parts.

- Two Big Ideas in React
  - Everything is a component: Component Hierarchy
  - JSX: Write code that resembles HTML

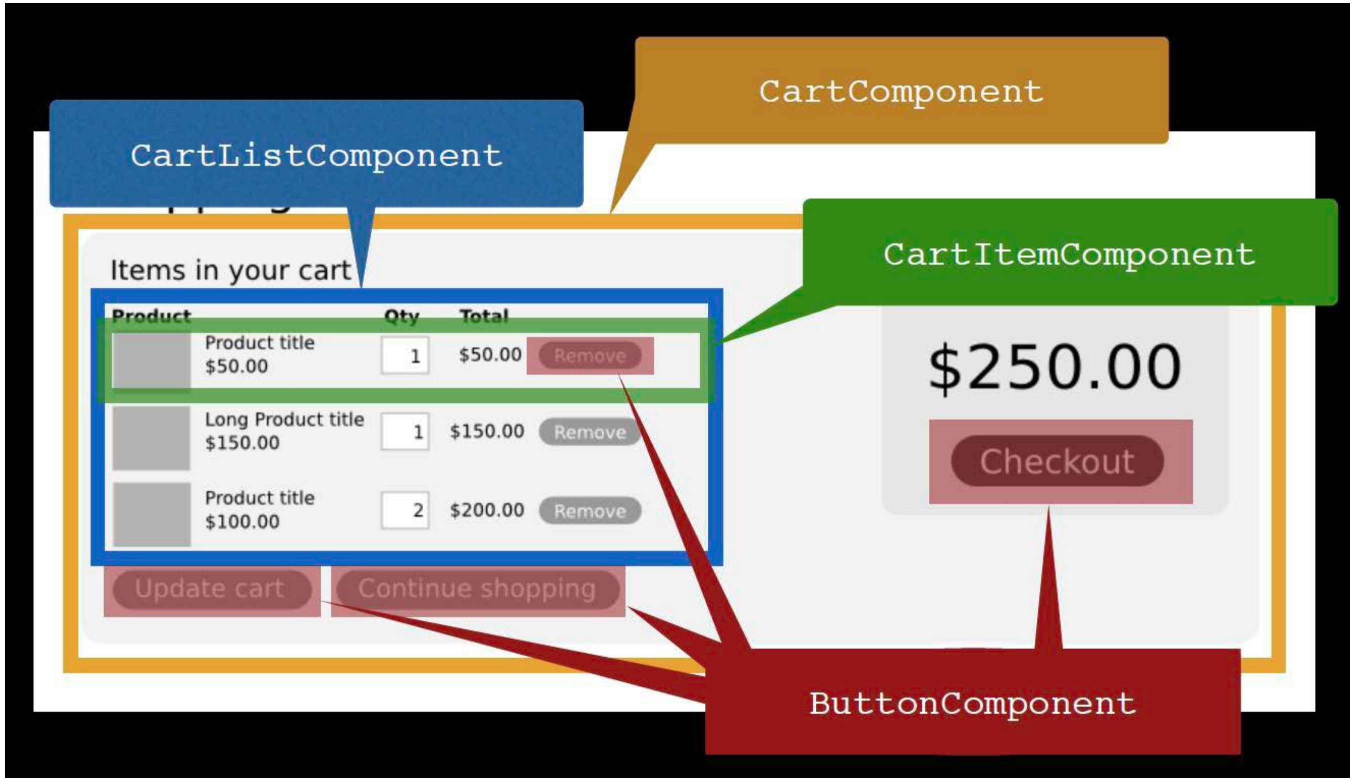
Component Hierarchy





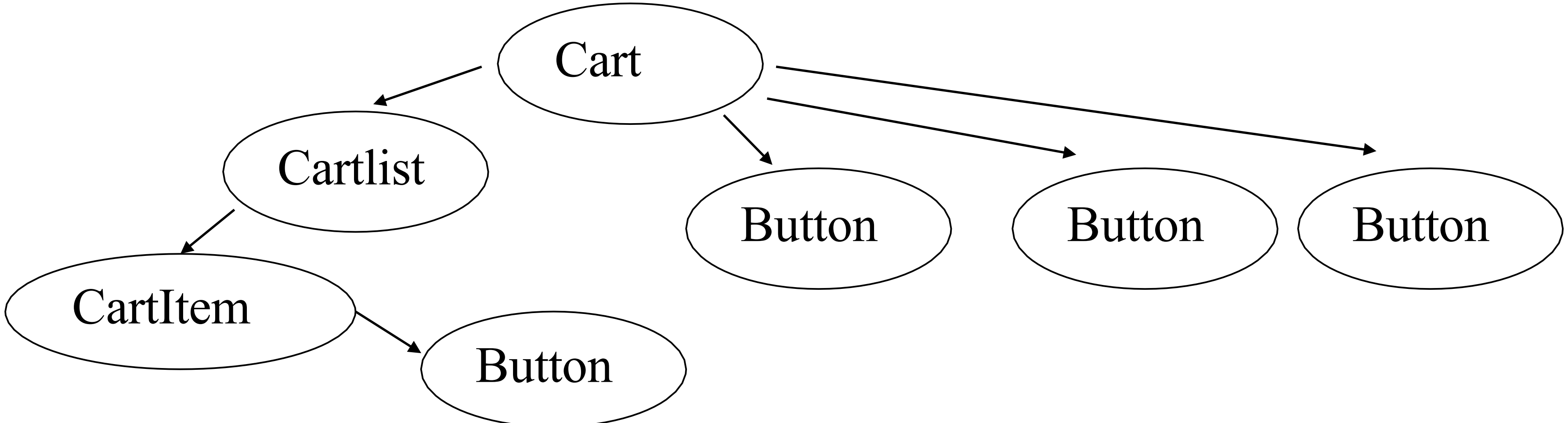
- Everything in React is a component.
- There is one root component
- There are other children components.
- One really powerful feature of react is that components can be reused for other applications.

Component Hierarchy (another example)



Items in your cart

Product		Qty	Total	
	Product title $50.00	1	$50.00	Remove
	Long Product title $150.00	1	$150.00	Remove
	Product title $100.00	2	$200.00	Remove



Why components are good?

- Composable
  - We can use components within each other.
- Reusable
  - Components can be reused for different applications.
- Maintainable
  - Easy to maintain since all the logic for the component are self contained
- Testable
  - We can test the correctness of each component individually.

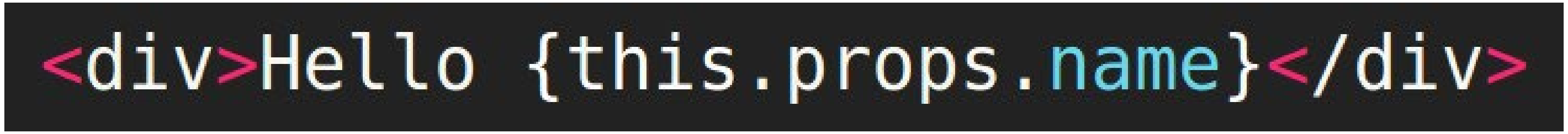
JSX

- React allows you to program in a really cool and intuitive programming language.
- The code has HTML-like syntax.
- JSX is optional.
- You can program in raw JavaScript if you want.

https://reactjs.org/docs/introducing-jsx.html

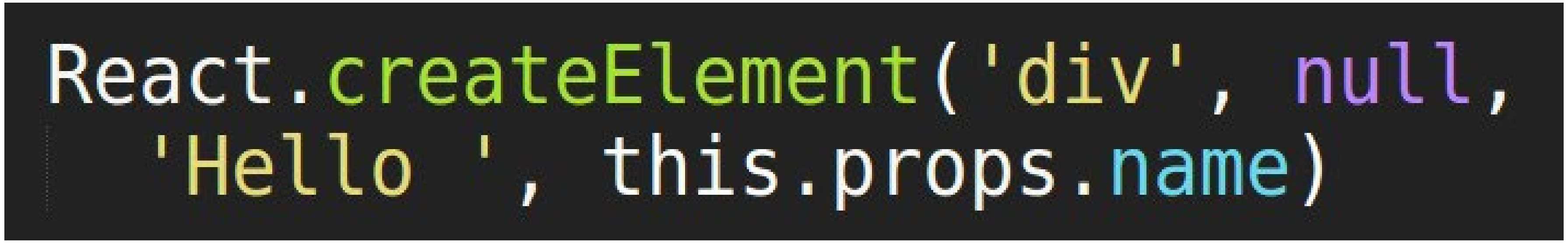
JSX versus Javascript

- For example the following JSX



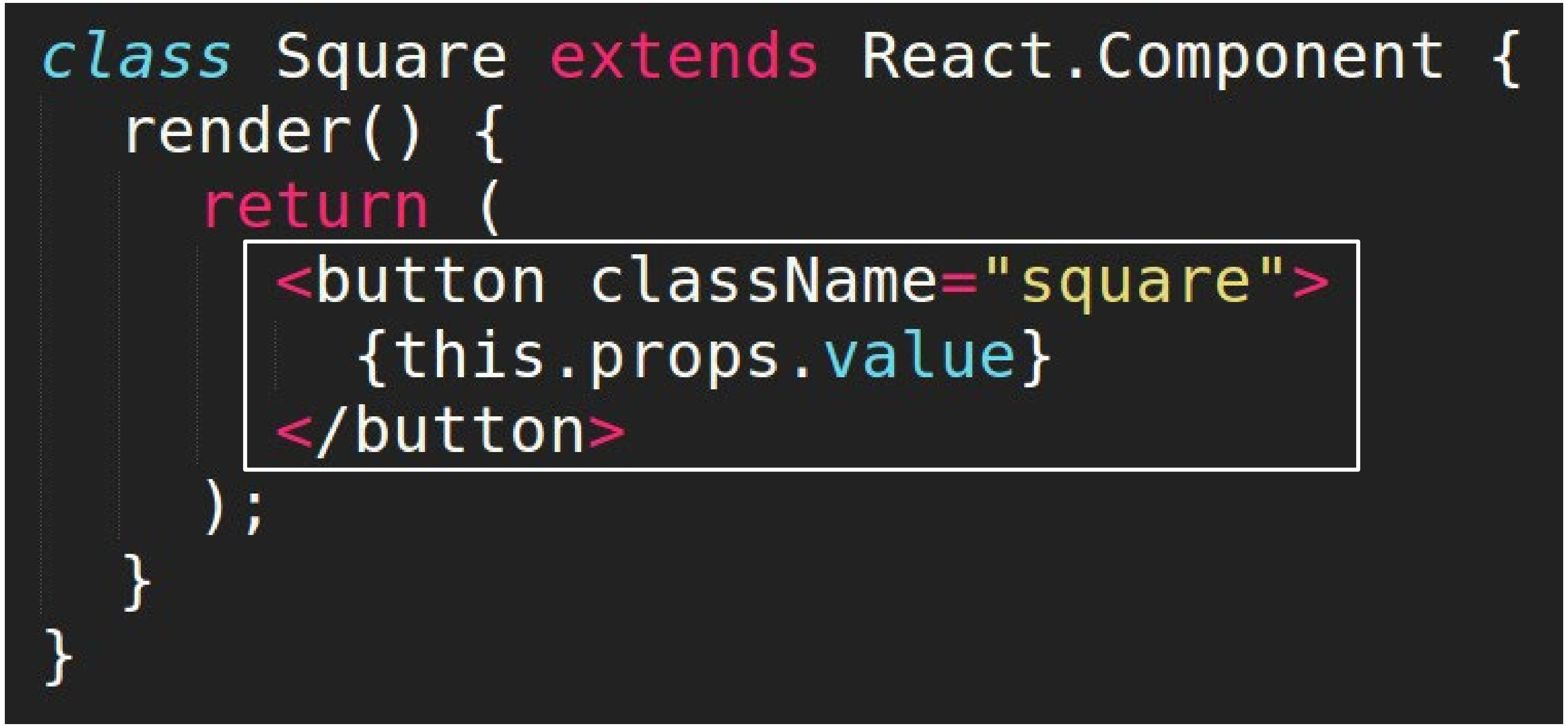
```
<div>Hello {this.props.name}</div>
```

- Looks like the following in JavaScript



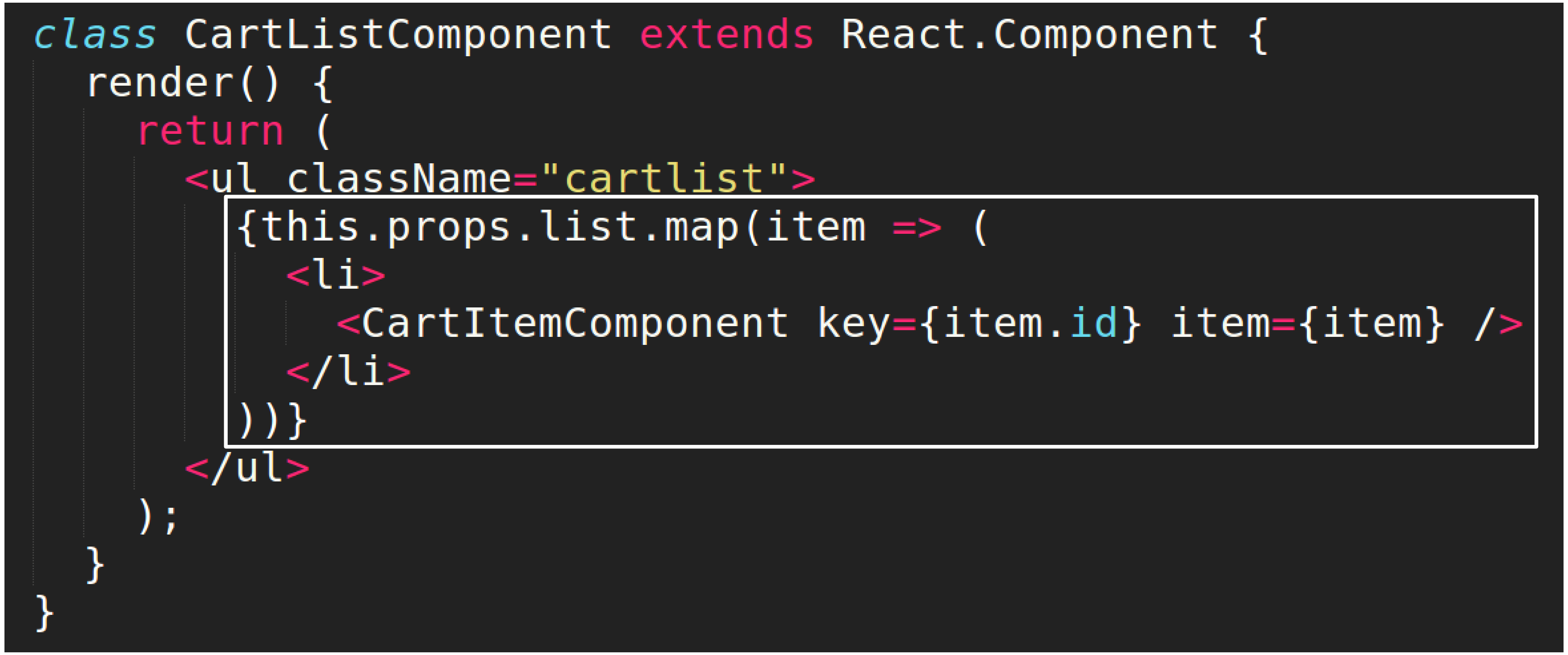
```
React.createElement('div', null,
  'Hello ', this.props.name)
```

Another JSX Example



```
class Square extends React.Component {
  render() {
    return (
      <button className="square">
        {this.props.value}
      </button>
    );
  }
}
```

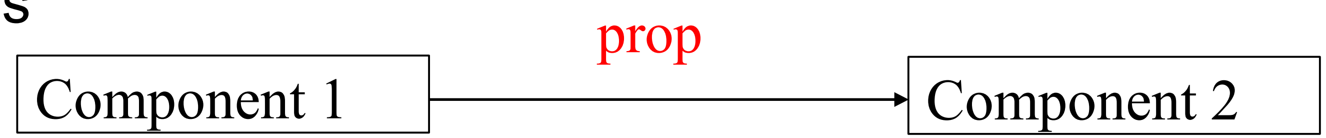
Another JSX Example
(Notice the use of JavaScript functions)



```
class CartListComponent extends React.Component {
  render() {
    return (
      <ul className="cartlist">
        {this.props.list.map(item => (
          <li>
            <CartItemComponent key={item.id} item={item} />
          </li>
        ))}
      </ul>
    );
  }
}
```

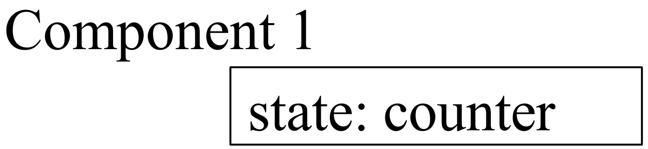
Passing and Storing Data

- In react data can be pass from component to component using props



  - You can think of props like parameters in a function
    - Calling function: displayName(25, “Peter”)
    - Function prototype: displayName(int age, string name)

- Components can store data in itself using state

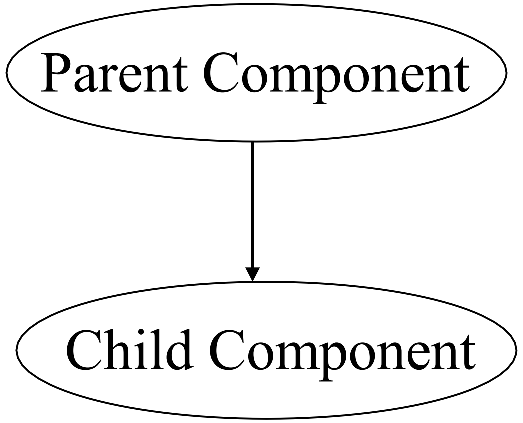


  - You can think of state like local variables in a function
    - Func() {

```
Func() {
   int counter;
   …..
}
```

React Properties

```
class ParentComponent extends React.Component {
  …
  render() {
    return (
      <div>
        <ChildComponent firstname="Zhen" />
      </div>
    );
  }
}
```



Data is passed from parent to child via props

```
class ChildComponent extends React.Component {
  …
  render() {
    return (<h1>Hello {this.props.firstname}</h1>);
  }
}
```

Props are immutable

- The following is not allowed:

```
class ChildComponent extends
React.Component {
 …
 doStuff
 () {
   this.props.firstname = “Peter”;
  }
}
```

- The child component can not change the prop.

Pass down properties via Spread Attributes

- Take the following example:
- <Comp1 foo={x} bar={y} choo={z} />
- In the above example we are passing down 3 properties to the Comp1 component. Another way to do this is via the spread attribute
- var myProps = {}; myProps.foo = x;
myProps.bar = y; myProps.choo = z;
- <Comp1 {…myProps}>
- We can override property (note order is important, later properties override earlier ones)
- <Comp1 {…myProps} foo={h}/>
- We can also add an additional property like the following:
- <Comp1 {…myProps} doo={a}/>

State is Mutable

```
class ChildComponent extends React.Component {
  // Initializes the active state variable to false
  constructor(props) {
    super(props);
    this.state = {active: false};
  }
  // Sets the active state variable to true
  makeActive() {
    this.setState({active : true});
  }
  // Renders the component
  render() {
    // Uses the active state variable
    return ( <input type="checkbox" checked={this.state.active} />);
  }
}
```

- Each component can have a state object. The state object can be mutated by the component.

State can become props

```
class ParentComponent extends React.Component {
  …
  render() {
    return (
      <div>
        <ChildComponent active={this.state.active} />
      </div>
    );
  }
}
```

```
class ChildComponent extends React.Component {
  …
  render() {
    // Uses the active state variable
    return (<input type="checkbox" checked={this.props.active} />);
  }
}
```

- The state object of a parent can be passed as a prop to a child.

Pure React Components

Standard React component	Pure React component

`class Foo extends React.Component`
`  { render() {`
`    return (`
`      <div>`
`        <p>{this.props.prop1}</p>`
`        <p>{this.props.prop2}</p>`
`      </div>`
`    );`
`  }`
`}`
	`// Version I: “Plain” JavaScript`
	`const Foo = function(props)`
	`  { return (`
	`    <div>`
	`      <p>{props.prop1}</p>`
	`      <p>{props.prop2}</p>`
	`    </div>`
	`  );`
	`};`

	`// Version II: Using ES6 features`
	`const Foo = ({prop1, prop2}) => (`
	`  <div>`
	`    <p>{prop1}</p>`
	`    <p>{prop2}</p>`
	`  </div>`
	`);`

- Pure react components is a convenience feature for writing simple components that only have a render function.
  - In the pure react component we do not need to explicitly specify the render function.
  - In the pure react component everything is in the render function.
- The arguments of the function are the just of a list of props passed in from the parent component.

Examples

- We will show sequence of examples starting from the basic hello world and then adding more and more features on.

Example 1 Basic Hello World

The HTML file

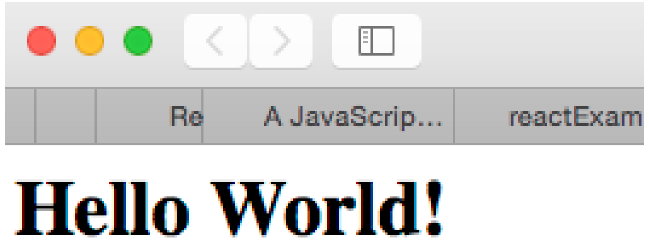
```
<meta name="robots" content="noindex">
<html>
<head>
  <meta charset="utf-8">

    <script src="https://unpkg.com/react@latest/dist/react.js"></script>
    <script  src="https://unpkg.com/react-dom@latest/dist/react-dom.js"></script>
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
</head>
<body>
       <div id="example"></div>
       <script src = labelapp.js type="text/babel">
</body>
</html>
```

labelapp.js File

- The code below prints out Hello World! in the web page.

```
class LabelApp extends React.Component {
    render() {
         return (<h1> Hello World! </h1>);
    }
}

// This function renders the LabelApp component at the location of
// the element that has id example (see previous slide).
ReactDOM.render(
        <LabelApp />,
        document.getElementById('example')
);
```

Output for Example 1



Hello World!

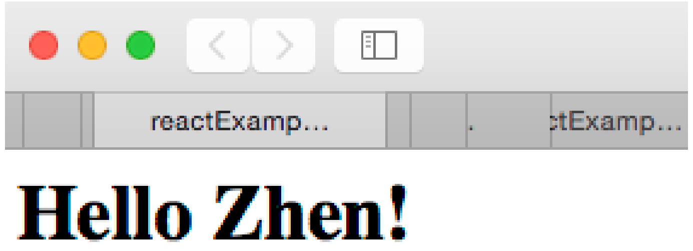
Example 2 Using Properties

Using properties

```
class LabelApp extends React.Component {
    render() {
        return (<h1> {this.props.message}</h1>);
    }
}

ReactDOM.render(
      <LabelApp message = "Hello Zhen!" />,
      document.getElementById('example')
);
```

- This example shows how to pass data from the root component to the LabelApp child component via the property message.

Output for Example 2



Hello Zhen!

Example 3 Using State

Using State

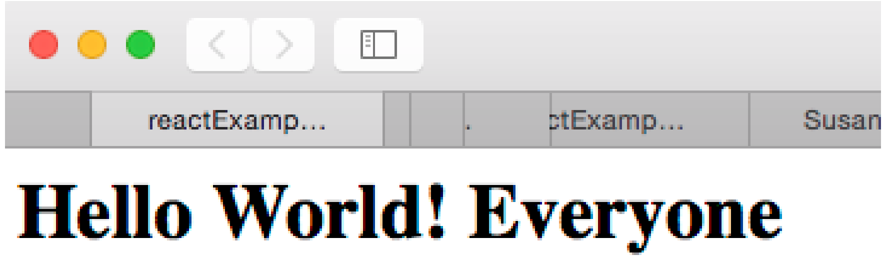
```
class LabelApp extends React.Component {
    static defaultProps =  {
        message: "Hello World!"
    }
    constructor(props) {
        super(props)
        this.state = {message: props.message, message2 : "Everyone"};
    }
    render() {
        return (<h1> {this.state.message} {this.state.message2} </h1>);
    }
}

ReactDOM.render(
     <LabelApp />,
     document.getElementById('example')
);
```

Component LabelApp

state: message
state: message2

- In this example there are two state variables: message and message2
  - The constructor function initializes the two state variables.
- There is also a property called message (this.props.message)
  - The defaultProps assignment sets a default value for the message property.
- The render function uses both of the state variables.

Output for Example 3



Hello World! Everyone

Example 4 Events

Events Example

```
class LabelApp extends React.Component {
    static defaultProps =  {
        message: "Hello World!"
    }
    constructor(props) {
        super(props)
        this.state = {message: props.message};
    }
    onClick = () => {
        this.setState({message: "Hello Zhen!"});
    }
    render() {
        return (<h1 onClick ={this.onClick} > {this.state.message} </h1>);
    }
}
```

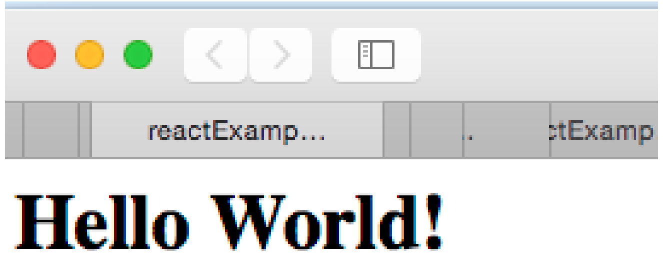
Event triggers function call

```
ReactDOM.render(
     <LabelApp />,
     document.getElementById('example')
);
```

- In the above example when the Hello World heading is clicked the message is replaced with Hello Zhen!
    - This is done by generating the onClick event which in turn calls the onClick function inside the LabelApp Component.

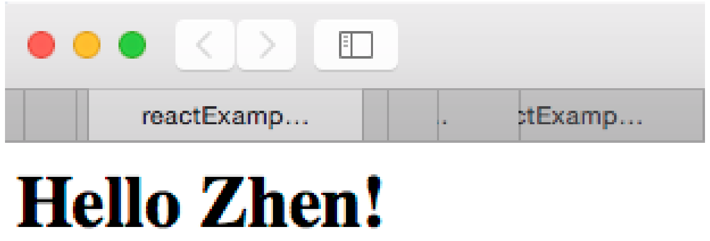
Output for Example 4

- Before clicking on Hello World!



Hello World!

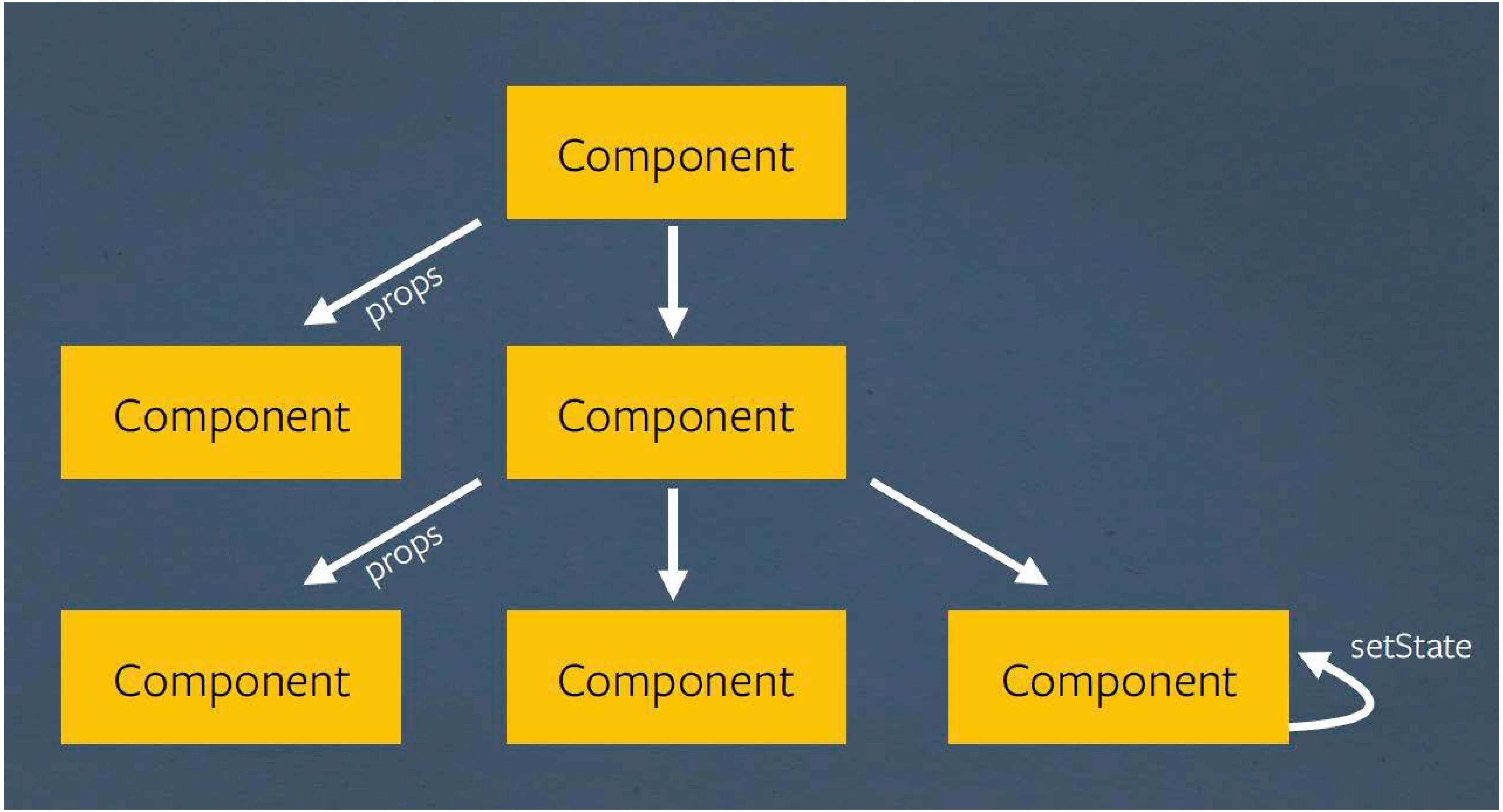
- After clicking on Hello World!



Hello Zhen!

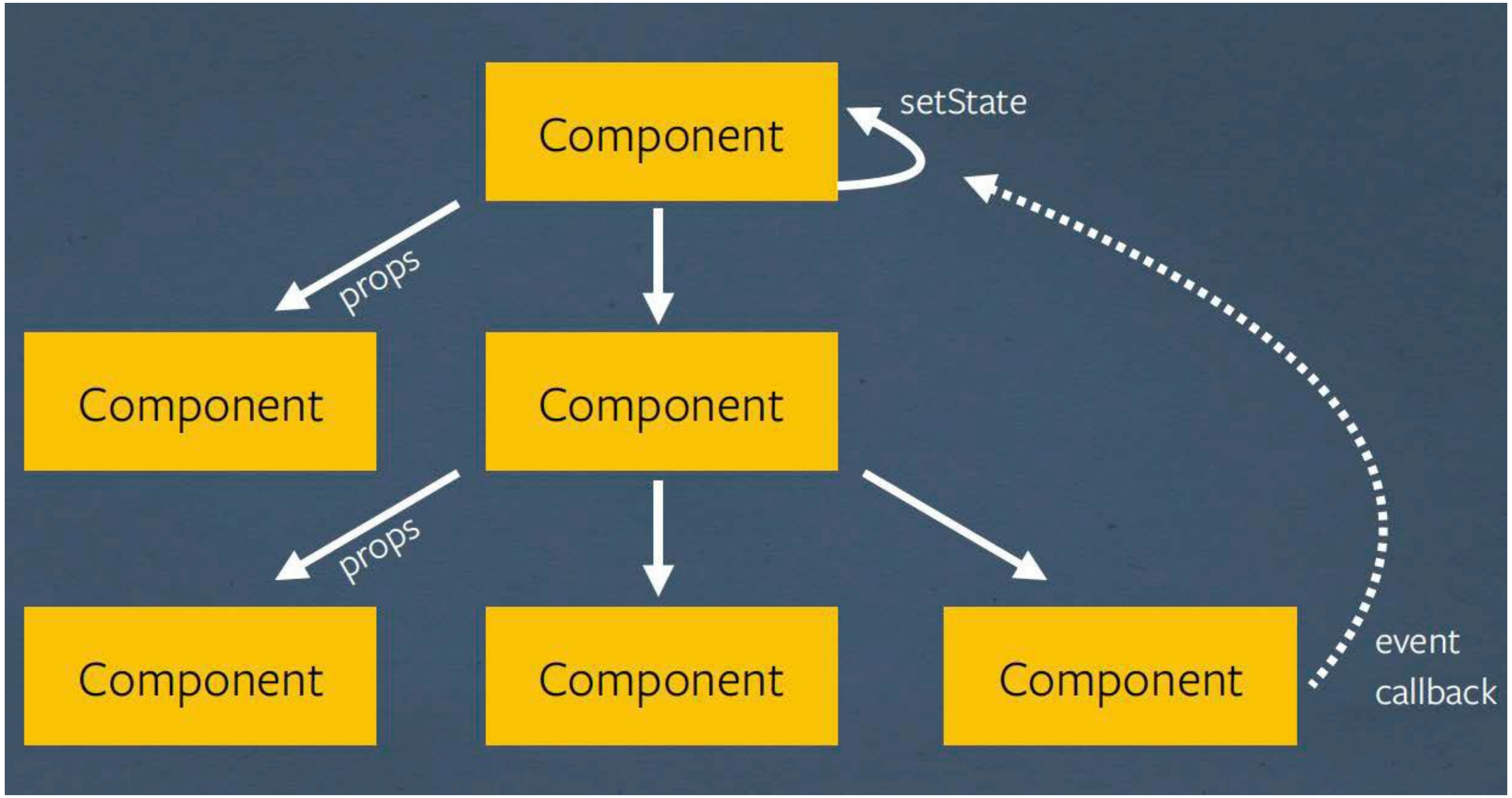
How Does Data Flow?

Data flows from Parent to Child via Props



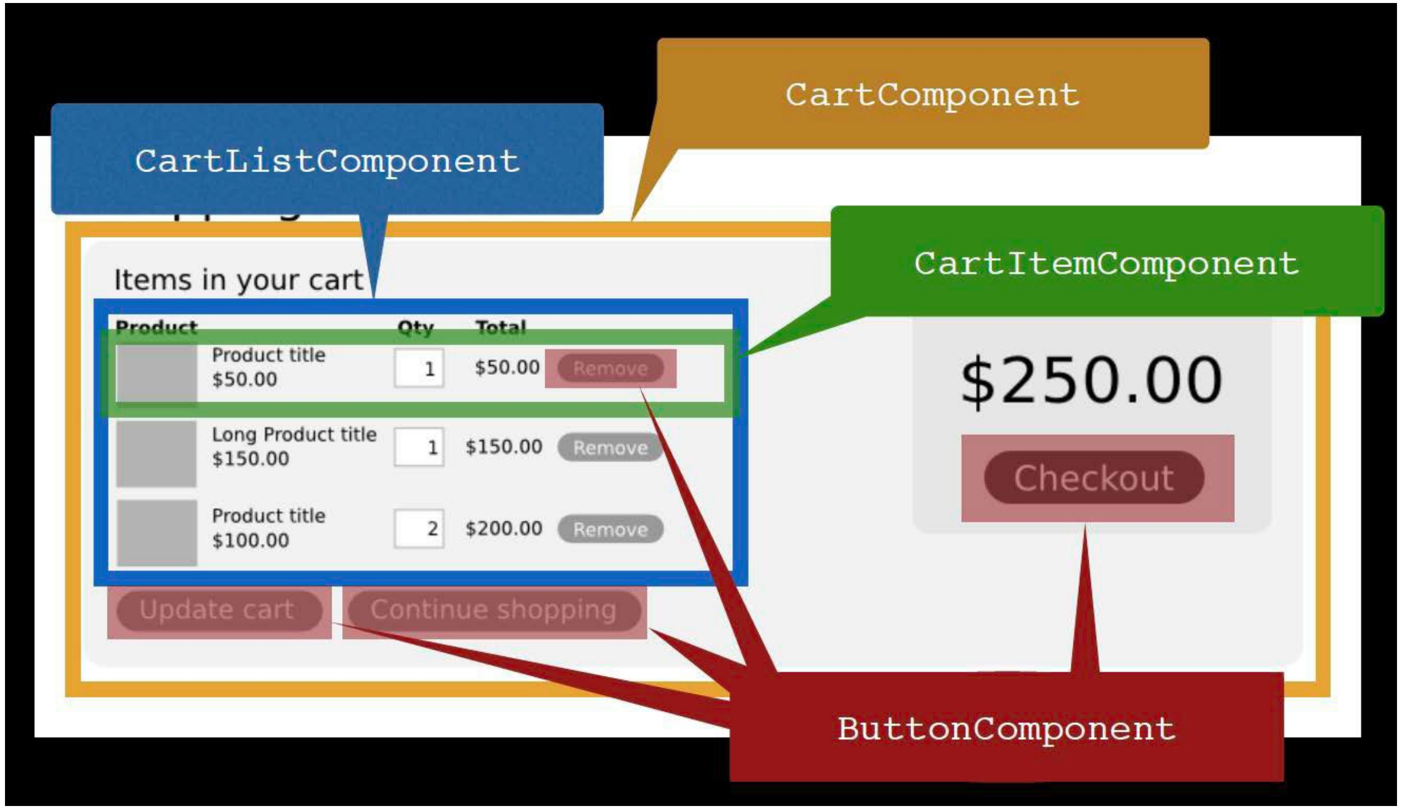
Events Flow Up

- When an event occurs the event is passed to a high level component.
- Example events are:
  - Someone enters text into a textbox
  - Someone clicks a button



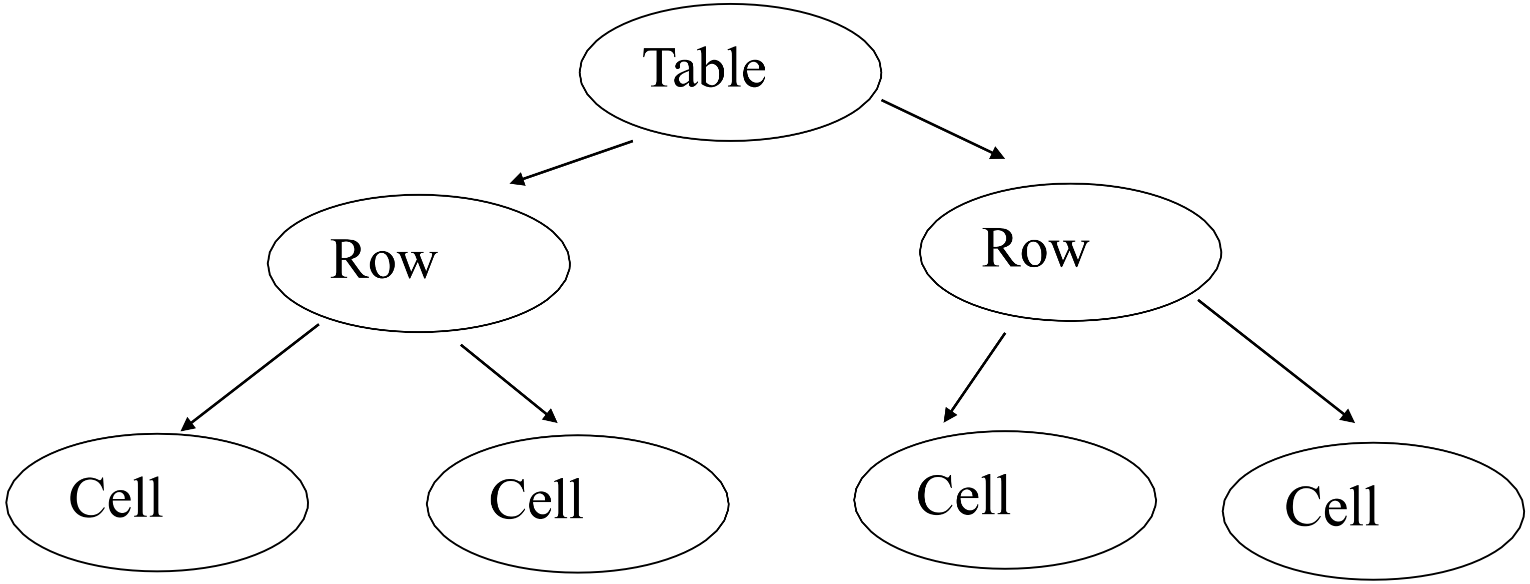
Where to store state?

- State needs to be stored as high up as needed to be passed down to affected children.
- In the example below the price of every product and quantity
- placed in the cart need to be stored in the parent CartComponent because all of this information is needed to compute the total shown inside the CartComponent.



Product	Qty	Total	
Product title $50.00	1	$50.00	Remove
Long Product title $150.00	1	$150.00	Remove
Product title $100.00	2	$200.00	Remove

The Table Example (Hierarchy of Components)



Table

Row

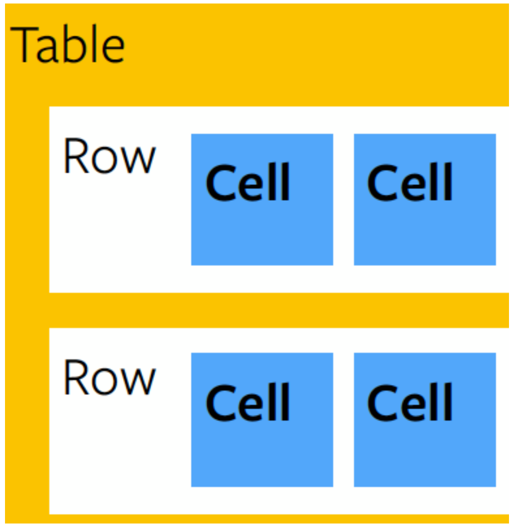
Row

Cell

Cell

Cell

Cell



Table

Row

Cell

Cell

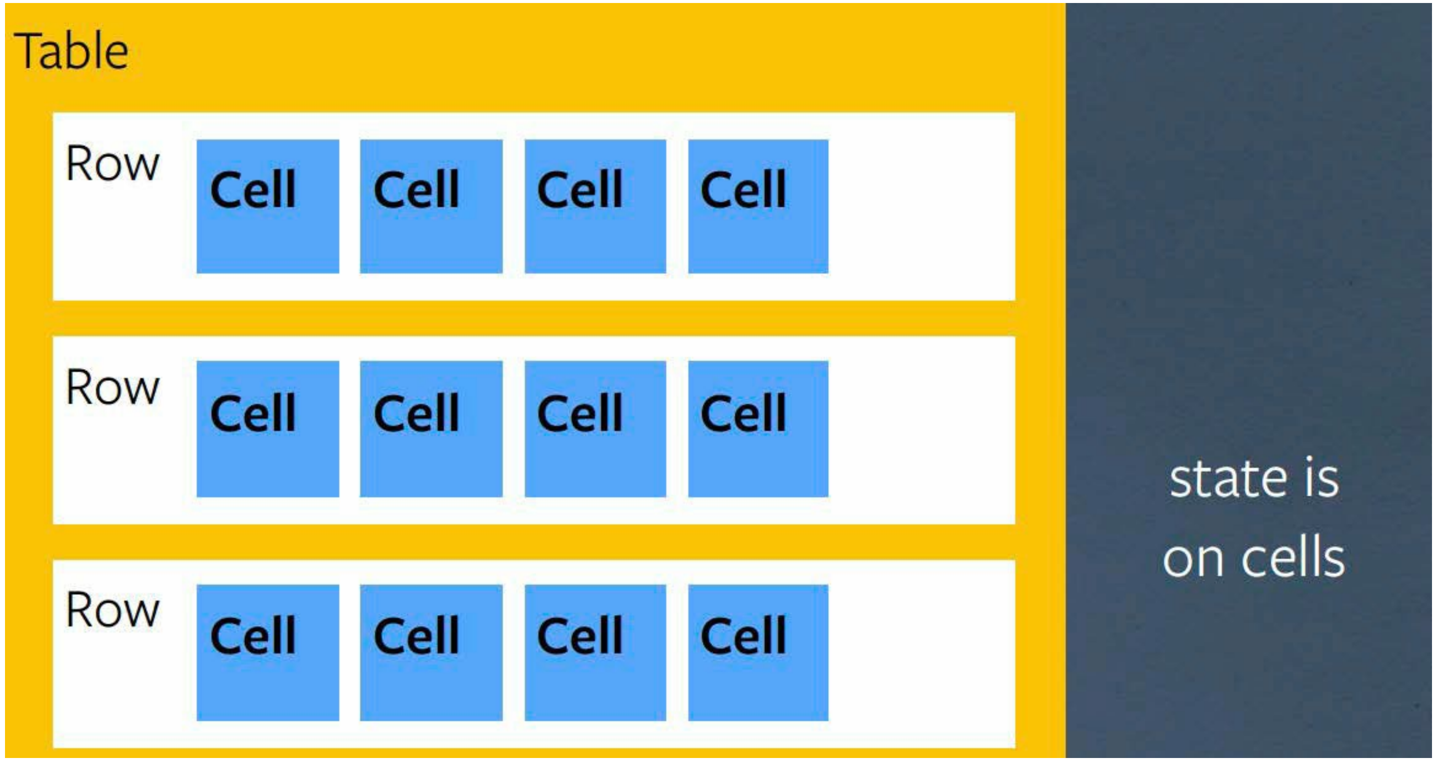
Row

Cell

Cell

The Table Example

- In this example, you can store the state within each cell component since the different components do not need to know each other’s state.



Table

Row Cell Cell Cell Cell

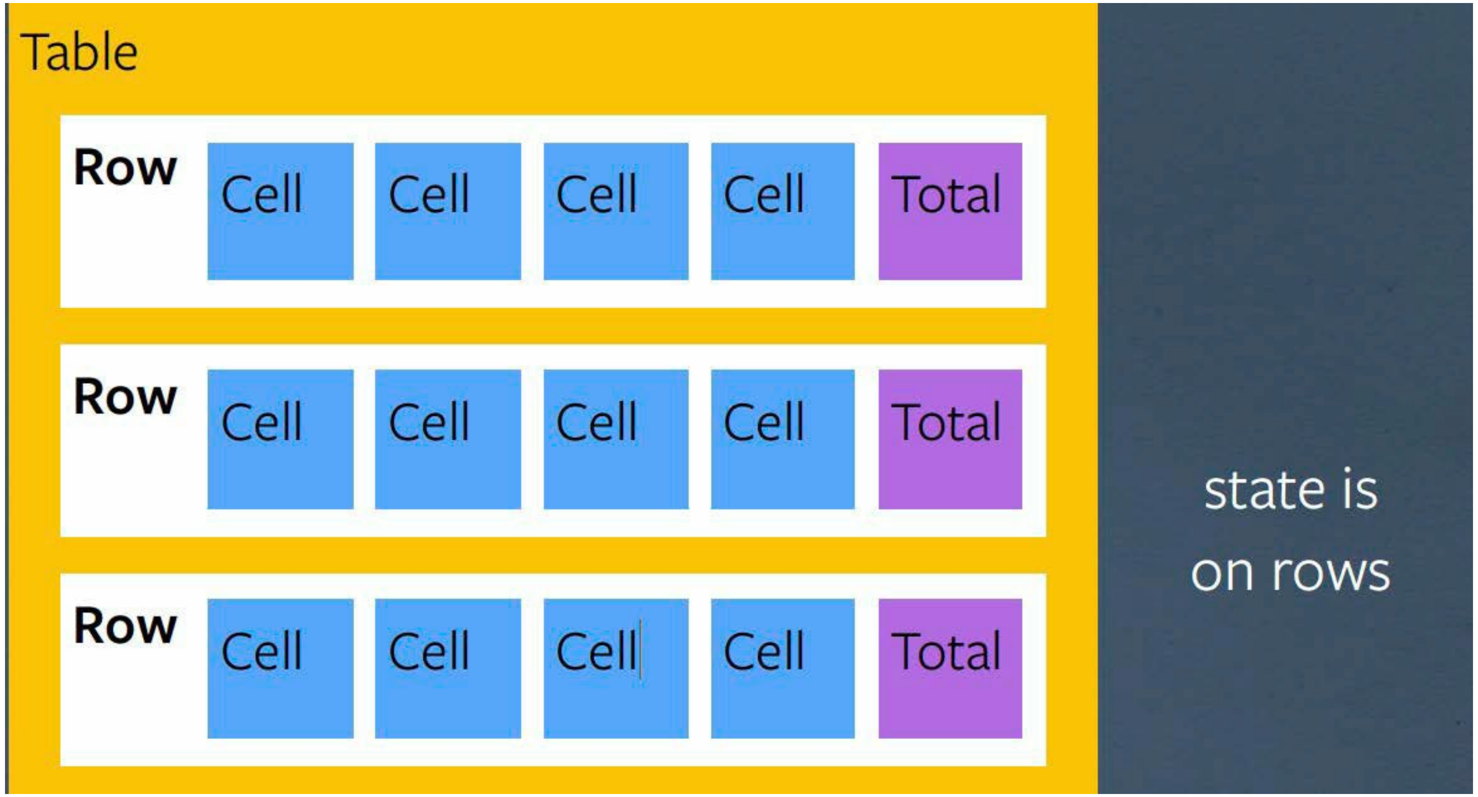
Row Cell Cell Cell Cell

Row Cell Cell Cell Cell

state is on cells

The Table Example

- In this example we need to generate a total for each row of the data.
- The row component is higher up the hierarchy and hence can not
- access the state of the cells.
- Hence the state for all the cells in this example need to be stored in the rows.



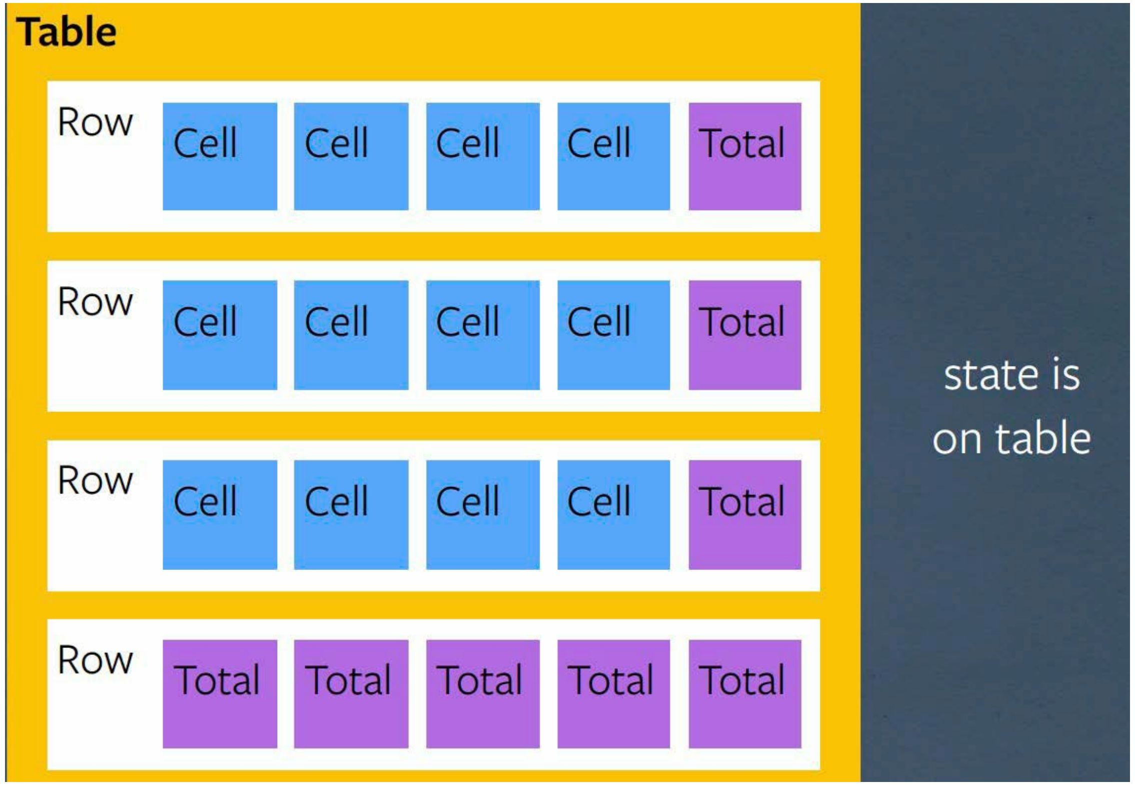
Table

Row	Cell	Cell	Cell	Cell	Total
Row	Cell	Cell	Cell	Cell	Total
Row	Cell	Cell	Cell	Cell	Total

state is on rows

The Table Example

- In the following example we need to store state in the root Table component since we need to sum across both rows and columns.



Table

Row	Cell	Cell	Cell	Cell	Total
Row	Cell	Cell	Cell	Cell	Total
Row	Cell	Cell	Cell	Cell	Total
Row	Total	Total	Total	Total	Total

state is on table

The Table Example (Table Component)

```
class Table extends React.Component {
        constructor(props) {
            super(props)
            const data = [];
            for (let r = 0; r < props.rows; r++) {
                data[r] = [];
                for (let c = 0; c < props.columns; c++) {
                    data[r][c] = 0;
                }
            }
            this.state = {data: data};
        }
```

- In the above example we store the state in the 2D array inside the Table Component.
    - We will then pass all the state information to the children components via properties.
- We initialize all the array values to 0.

The Table Example
(Table Component Render Function)

```
render() {
    const rowArray = [];
    const columnTotals = [];
    for (let c = 0; c < this.props.columns; c++) {
         columnTotals[c] = 0;
    }
    for (let r = 0; r < this.props.rows; r++) {

         let total = 0;
         const childrenArray = [];
         for (var c = 0; c < this.props.columns; c++) {
              childrenArray.push(<Cell row={r} column={c}
                         value={this.state.data[r][c]}
                         onChange={this.onCellChange} />);
              total += this.state.data[r][c];
              columnTotals[c] += this.state.data[r][c];
        }
         childrenArray.push(<Total value={total} />);
         rowArray.push(<Row children = {childrenArray} />);
```

… (continued on next page) …

- We first create an array of rows and an array storing the totals for each column.
- For each row we first create all the cells that go inside the row.
  - We use properties to pass the row number, column number and value.
  - We also compute the row totals and column totals.
- We put the row total into the children array
- Next we create a row component and place the array of cells inside it.

The Table Example (Table Component Continued)

```
render: function() {

    … (continued from previous page) …

    const totalsArrayComponent = []
    let finalTotal =0;
    // Compute the column totals and put them into the final row which stores all the column
    // totals.
    for (let c = 0; c < this.props.columns; c++) {
        totalsArrayComponent[c] = <Total value={columnTotals[c]} />
        finalTotal += columnTotals[c];
    }
    totalsArrayComponent.push(<Total  value= {finalTotal}/>);
    rowArray.push(totalsArrayComponent);
    // Finally return the array of rows enclosed inside the <table> HTML tag.
    return (<table>   {rowArray} </table>)}

// Render Table component and pass in the properties columns and rows.
ReactDOM.render(
    <Table columns={4} rows = {5}/>,
   document.getElementById('example'));
)
```

The Table Example (Row and Total Components)

```
class Row extends React.Component {
    render() {
        return (<tr>{this.props.children}</tr>);
    }
};

class Total extends React.Component {
    render() {
        return (<th>{this.props.value}</th>);
    }
};
```

Table Example (Events)

```
class Cell extends React.Component {
    onChange = (e) => {
        this.props.onChange(
            this.props.row, this.props.column, parseInt(e.target.value)
        );
    }

    render() {
        return (<td> <input type="number" value={this.props.value}
                onChange={this.onChange} />  </td>);
    }
};
class Table extends React.Component {
     ….
     onCellChange = (row, column, value) => {
                    this.state.data[row][column] = value;
                    this.setState({data: this.state.data});
     }
     Render() {
            ….
            childrenArray.push(<Cell row={r} column={c}   value={this.state.data[r][c]}
                                 onChange={this.onCellChange} />);
            …
     }
```

the new changed value passed in from the textbox via event e

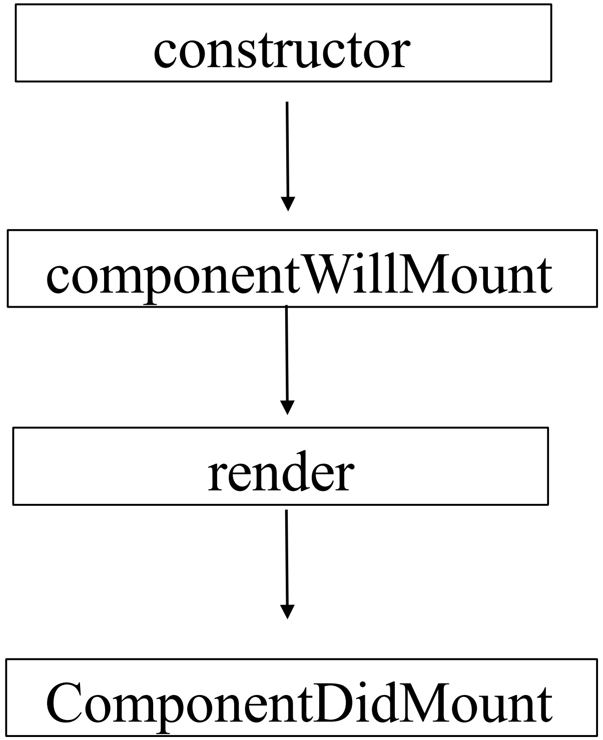
Here is where you detect the change.
The code then calls the onChange function

Here the onCellChange function changes the state of the 2D array to include the updated value.

Calls onCellChange function in child component via onChange prop

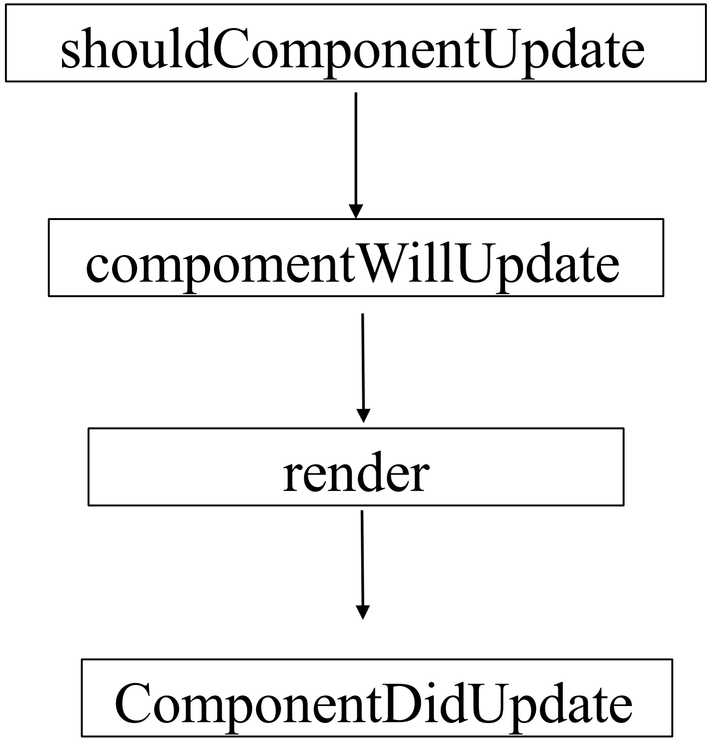
Lets Try it out!

React Lifecycle (Initial Render)



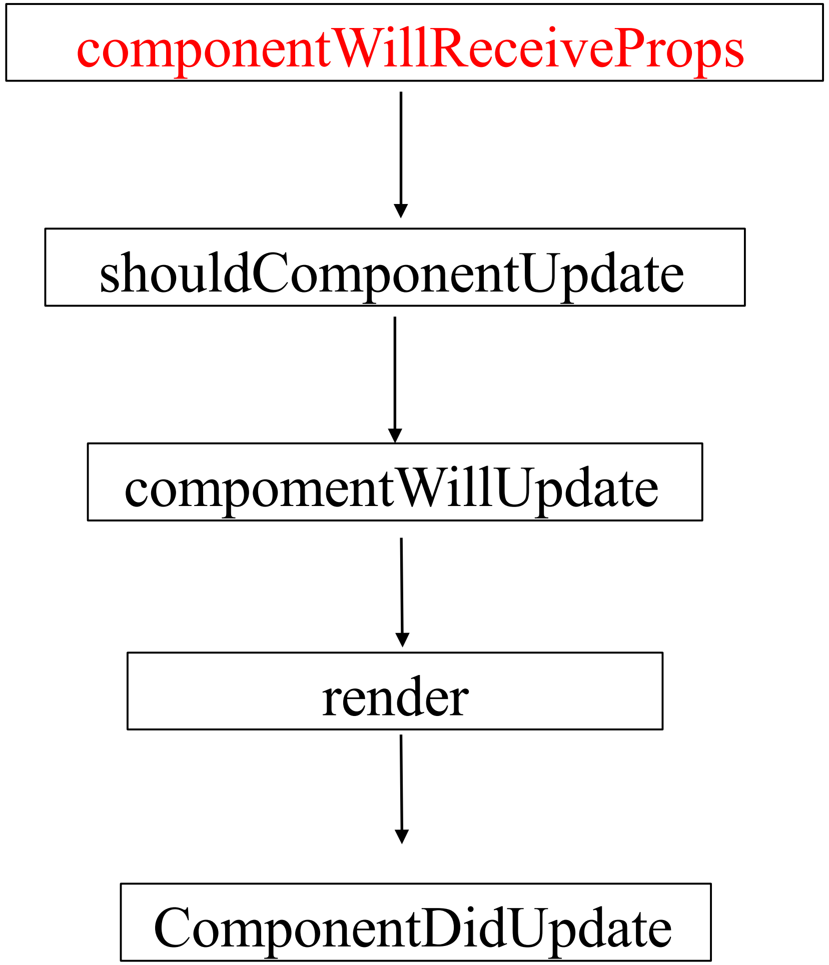
- This shows the order by which functions are called in the React Lifecycle.
- Mount means initial render.
- The constructor function is usually used to initialize state variables to a specified value. You don't need to have a constructor.
- The componentWillMount function allows you to insert code that is run before the initial render (mount).
- The componentDidMount function allows you to insert code that is run after the initial render.

React Lifecycle (change of state)



- The shouldComponentUpdate function allows you to insert code that decides whether the page needs to be rerendered as a result of a state change. So if you know no rerender is needed then you can use this function.
- The componentWillUpdate function allows you to insert code that is run just before the page gets rerendered as a result of state change.
- The componentDidUpdate function allows you to insert code that is run just after the page gets rerendered as a result of state change.

React Lifecycle (property changes)



componentWillReceiveProps

shouldComponentUpdate

compomentWillUpdate

render

ComponentDidUpdate

- The componentWillReceiveProps function is executed when the component will be receiving props. This function is executed on the first prop change but never on the first render.

Conclusion

- There are many different JavaScript Libraries/Frameworks.
- The react library is particularly useful for complex web pages which many different event handers.
- The declarative programming style of the react library simplifying the development complex web pages.