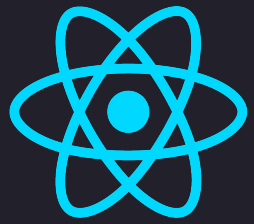
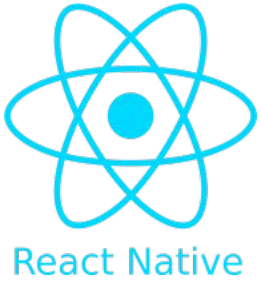


# React

A JavaScript library for building user interfaces


React


React


React Native


SVELTE

**The**
**Model View Controller (MVC or MTV)**

| | Django | Django & React |
|---|---|---|
| Model | Models.py | models.py |
| View | Templates | React |
| Controller | Views.py | views.py |


## Libraries Needed

React
- The main react library

ReactDom
- A package that provides DOM-specific methods that can be used at the top level of your app and as an escape hatch to get outside of the React model if you need to. Most of your components should not need to use this module.
  - `render` method

React-scripts (optional but highly recommended)
- A package that provides default scripts for React and also includes webpack and babel.



Babel

- Javascript compiler that converts ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.


Webpack

- Webpack is a build tool that bundles all of your assets, including Javascript, images, fonts, and CSS, in a *bundle* file(s) that executes the actual code in the browser.

**Note: Installing 'react-scripts' library manages webpack and babel for you so you don't have to create a babel or webpack config file.**

## Examples

.babelrc

```
{
 presets: ['react', 'env', 'stage-2', 'es2015'],
 plugins: ['transform-class-properties']
}
```

webpack.config.js

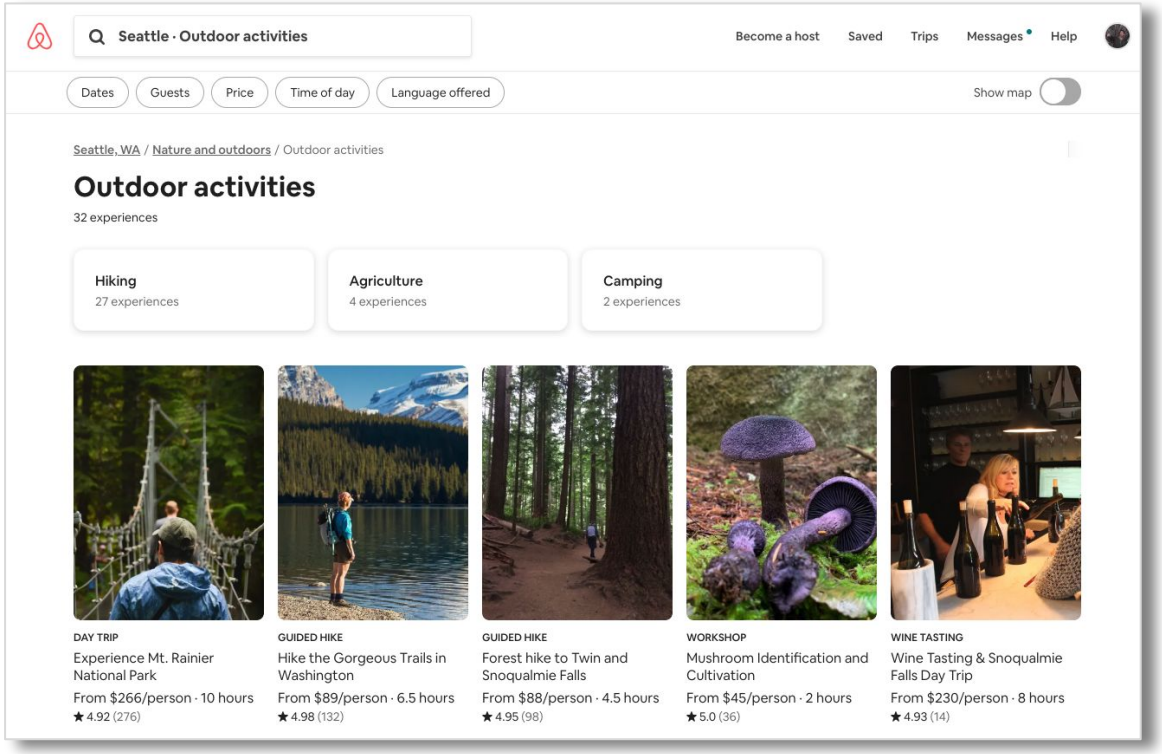
```
const isDev = process.env.NODE_ENV === 'development'

module.exports = {
 mode: isDev ? 'development' : 'production',
 entry: ['babel-polyfill', './client/index.js'],
 output: {
   path: __dirname,
   filename: './public/bundle.js'
 },
 devtool: 'source-map',
 module: {
   rules: [
     {
       test: /\.jsx?$/,
       exclude: /(node_modules|bower_components)/,
       loader: 'babel-loader',
     },
     {
       test: /\.css$/,
       use: ['style-loader', 'css-loader']
     }
   ]
 },
}
```


**`npx create-react-app`** `<appName>`

The `create-react-app` command creates a single page application using react and builds out the default react file structure and installs all react dependencies to get started. Similar to `django-admin startproject <project_name>`

# React




Components

- "Components let you split the UI into independent, **reusable pieces**, and think about each piece in isolation."
- Classed based components
- Functional based components
- Can have data as `state` and/or `props`

Props

- Components can accept data as `props`. Similar to passing arguments to a functior

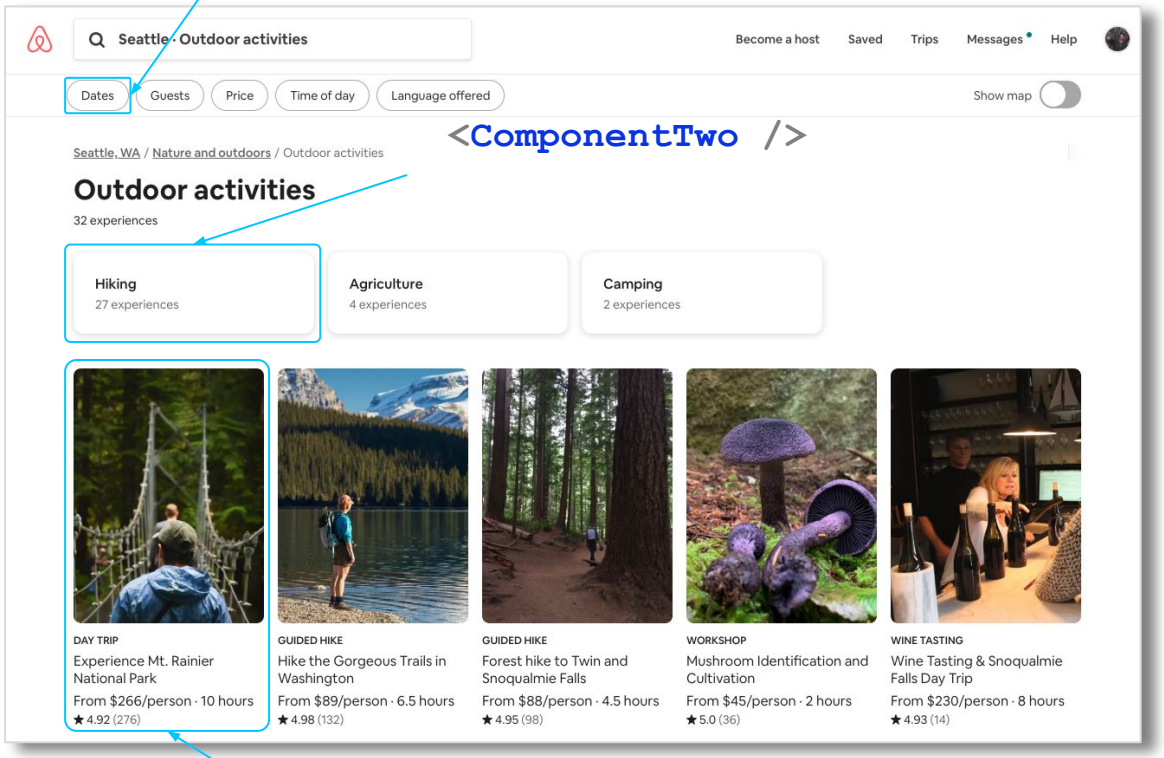
State

- `state` is an object that stores data that is local to the component
- Similar to `props` but is only local to the specific component

JSX

- Looks like and acts like HTML but is actually a syntax extension to JavaScript
- Allows us to write JavaScript in-line with 'html'
- Like jinja2 in python

# React




## Components

- "Components let you split the UI into independent, **reusable pieces**, and think about each piece in isolation."
- Classed based components
- Functional based components
- Can have data as `state` and/or `props`

## Props

- Components can accept data as `props`. Similar to passing arguments to a function.

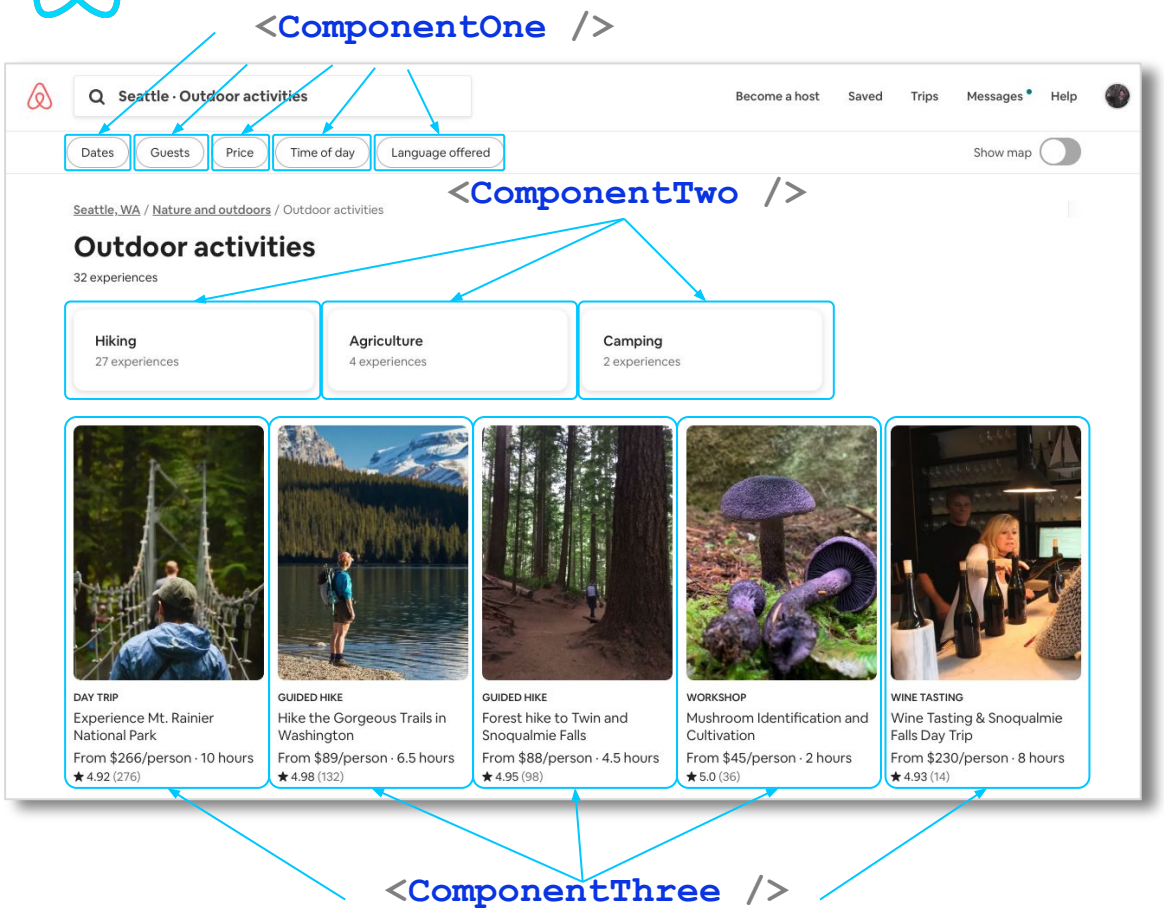
## State

- `state` is an object that stores data that is local to the component
- Similar to `props` but is only local to the specific component

## JSX

- Looks like and acts like HTML but is actually a syntax extension to JavaScript
- Allows us to write JavaScript in-line with 'html'
- Like jinja2 in python

# React




## Components

- "Components let you split the UI into independent, **reusable pieces**, and think about each piece in isolation."
- Classed based components
- Functional based components
- Can have data as `state` and/or `props`

## Props

- Components can accept data as `props`. Similar to passing arguments to a function.

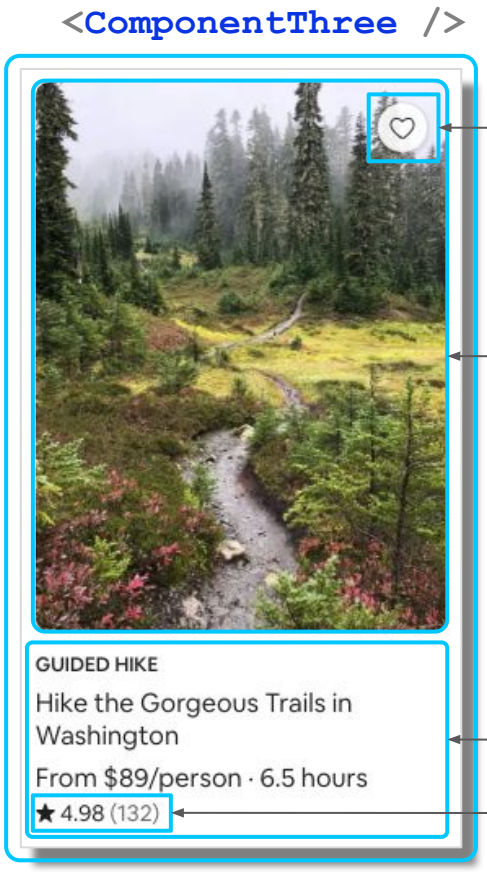
## State

- `state` is an object that stores data that is local to the component
- Similar to `props` but is only local to the specific component

## JSX

- Looks like and acts like HTML but is actually a syntax extension to JavaScript
- Allows us to write JavaScript in-line with 'html'
- Like jinja2 in python

# React




Components

- "Components let you split the UI into independent, **reusable pieces**, and think about each piece in isolation."
- Classed based components
- Functional based components
- Can have data as `state` and/or `props`

Props

- Components can accept data as `props`. Similar to passing arguments to a function.

State

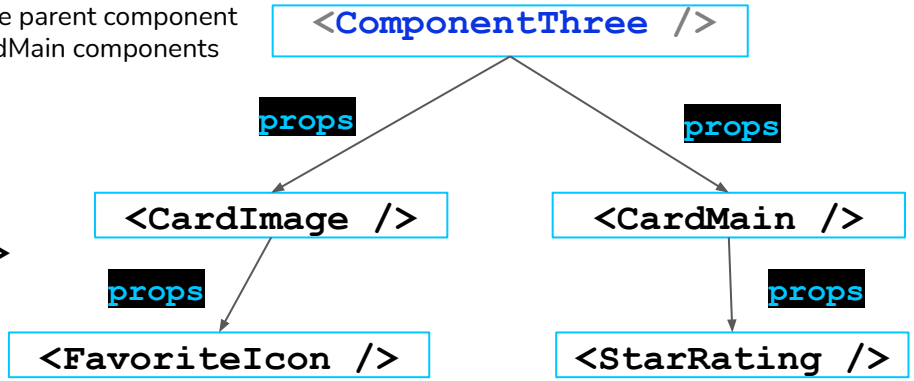
- `state` is an object that stores data that is local to the component
- Similar to `props` but is only local to the specific component

JSX

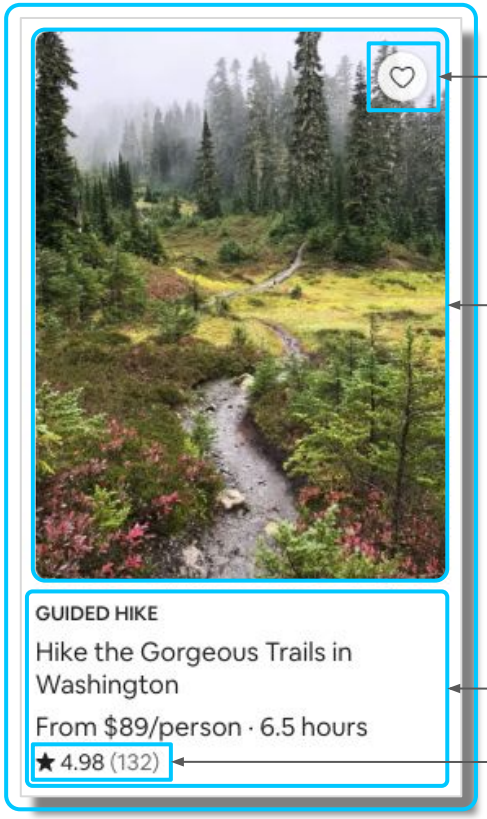
- Looks like and acts like HTML but is actually a syntax extension to JavaScript
- Allows us to write JavaScript in-line with 'html'
- Like jinja2 in python

# React

ComponentThree is the parent component of CardImage and CardMain components




<ComponentThree />




<FavoriteIcon />

<CardImage />

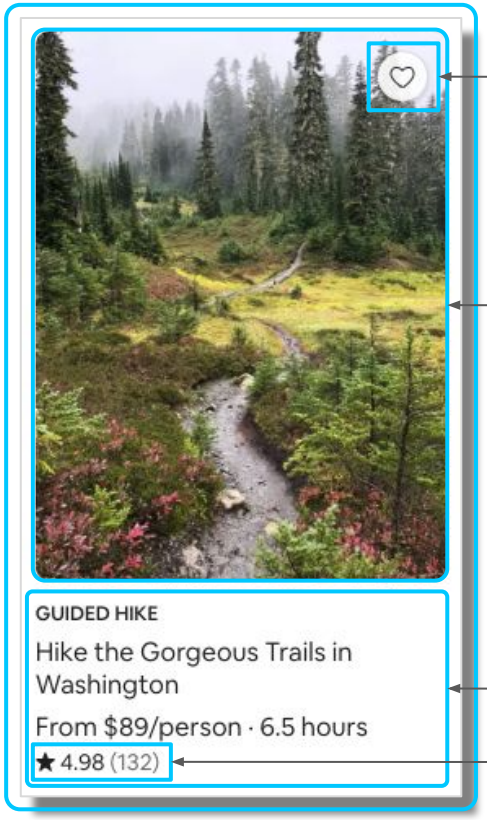
<CardMain />

<StarRating />

**props** is passed down from Parent Component to Child Component

# React

<ComponentThree />



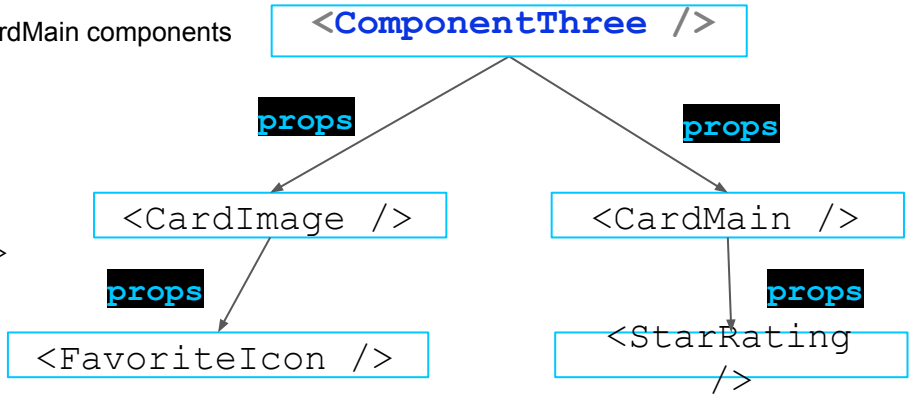

<FavoriteIcon />

<CardImage />

<CardMain />

<StarRating />

Parent of CardImage and CardMain components




```
15  // ComponentThree
16  return (
17    <div>
18      <CardImage />
19      <CardMain />
20    </div>
21  )
```

```
23  // CardImage
24  return (
25    <div>
26      <FavoriteIcon />
27      Some more elements like an image
28    </div>
29  )
```

```
31  // CardMain
32  return (
33    <div>
34      Some more elements like title, description, price, hours
35      <StarRating />
36    </div>
37  )
```

**props** is passed down from Parent Component to Child Component


There are two ways to create a component

1. Class Based Components
2. Functional Based Components

```jsx
import React, { Component } from 'react';

class ButtonComponent extends Component {
 // Declaring state in a class based component
 state = {
   count: 0
 };

 handleButtonClick = () => {
   this.setState({
     count: this.state.count + 1
   });
 };

 render() {
   return (
     <div>
       <h1> I have been clicked { this.state.count } times </h1>
       <button onClick={ this.handleButtonClick }> Click me! </button>
     </div>
   );
 }
}

export default ButtonComponent;
```

```jsx
import React, { useState } from 'react';

const ButtonComponent = () => {
 // Declaring state in a functional based component
 const [count, setCount] = useState(0)

 const handleButtonClick = () => {
   setCount(prevCount => {
     prevCount - 1
   })
 };

   return (
     <div>
       <h1> I have been clicked { count } times </h1>
       <button onClick={ handleButtonClick }> Click me!
</button>
     </div>
   );
}
export default ButtonComponent;
```

React

# A Component's Lifecycle


When a component renders to the DOM for the first time (document object model)

When a component changes state (data) or props (data) the component may/may not re-render

When a component is removed from the DOM


# Lifecycle Methods

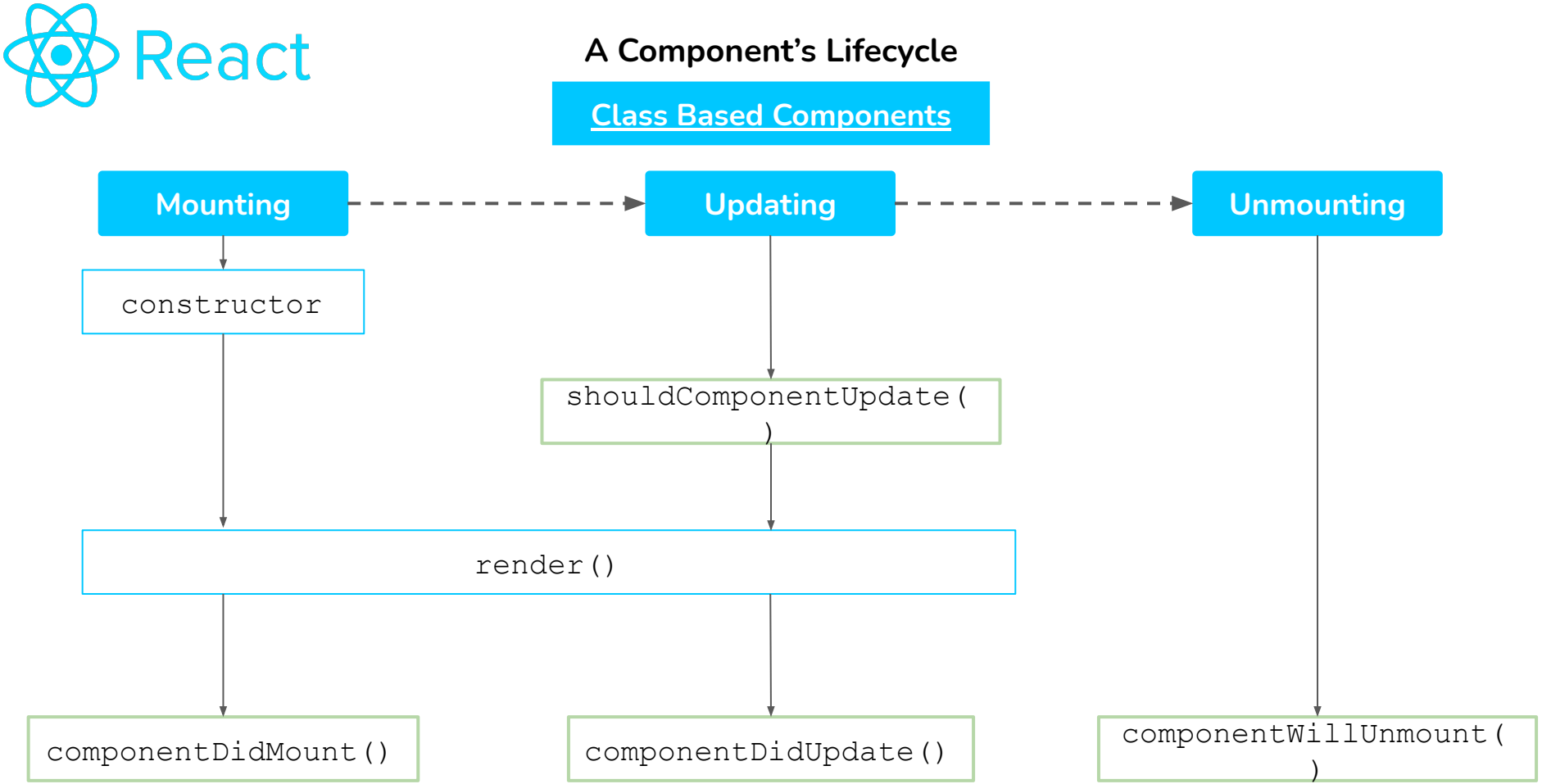
**Each class component has several “lifecycle methods” that you can override to run code at particular times in the process.**

```
componentDidMount()
```

```
shouldComponentUpdate()
```

```
componentDidUpdate()
```

```
componentWillUnmount()
```


React
A Component's Lifecycle
Class Based Components
Mounting
Updating
Unmounting
constructor
shouldComponentUpdate( )
render()
componentDidMount()
componentDidUpdate()
componentWillUnmount( )


# A Component's Lifecycle

**Hooks allow you to interact with data at different points in a component's lifecycle without using Class Based Components. You can now access data (state or props) using functional components.**

| Lifecycle Methods | Hook equivalent |
| --- | --- |
| `componentDidMount()` | |
| `shouldComponentUpdate()` | |
| `componentDidUpdate()` | `useEffect()` |
| `componentWillUnmount()` | |