

REACT LIFECYCLE

IT'S THE CI-RCLE...

TRAJECTORY

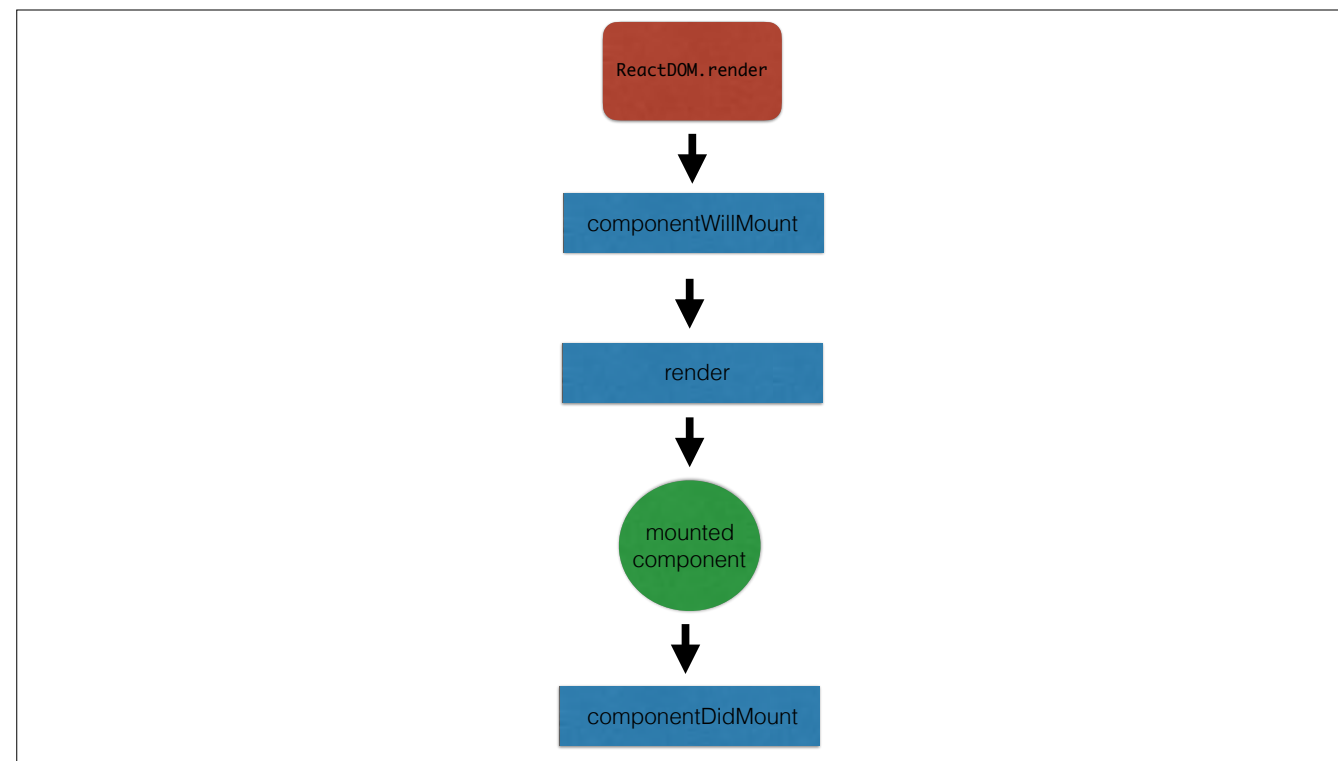
- **Basic React lifecycle methods**
- **Incorporating AJAX and other side-effects into React applications**

WHAT IS “LIFECYCLE”

- **When we render a component, React components go through several different stages in addition to the “render” stage**
- **React exposes the ability to “hook” into these stages so that we can perform certain actions ourselves**
 - Kind of like adding an event listener

Right now, we think of React components as simple state machines. We render them with some initial state. We then `setState`, and then they poop out a newly calculated view. `setState`, new view. `setState`, new view. And for the most part, that’s good, but there are a few other things that happen under the hood as well.

THE INITIAL RENDER



COMPONENT WILL MOUNT

- Happens before rendering
- Mostly used when using React to render strings on the server - don't use it when using client-side React

RENDER

- **This is when the component's render method is invoked**
 - Or when the functional component itself is invoked
- **React compares the JSX output of the render method with its internal “virtual DOM”, and makes a decision about how to update the actual DOM in a performant way**

VIRTUAL DOM

- **Just a big JS object representing the DOM tree**
 - internal, used by React when you render
- **Theory: manipulating the actual DOM is more expensive computationally than doing a little bit of JS**
- **So, React does a little bit of JS first so that it can do as little manipulation of the actual DOM as possible**

“MOUNTING”

- When the JSX your component represents gets turned into real, live DOM nodes by React, your component is said to be “mounted”

COMPONENT DID MOUNT

- Fires after the *initial* rendering
 - Does *not* fire on subsequent renderings caused by `setState`
- A great place to perform **AJAX requests to fetch data from your server**
- A great place to attach event listeners to non-React elements (ex. `window.addEventListener('scroll')`)

Any guesses on why this would be better than `componentWillMount`?

Using `DidMount` makes it clear that data won't be loaded until after the initial render. This reminds you to set up initial state properly, so you don't end up with undefined state that causes errors.

If you ever need to render your app on the server (SSR/isomorphic/other buzzwords), `componentWillMount` will actually be called twice – once on the server, and again on the client – which is probably not what you want. Putting the data loading code in `componentDidMount` will ensure that data is only fetched from the client.

```
class Blog extends React.Component {
  constructor () {
    super()
    this.state = {
      posts: []
    }
  }

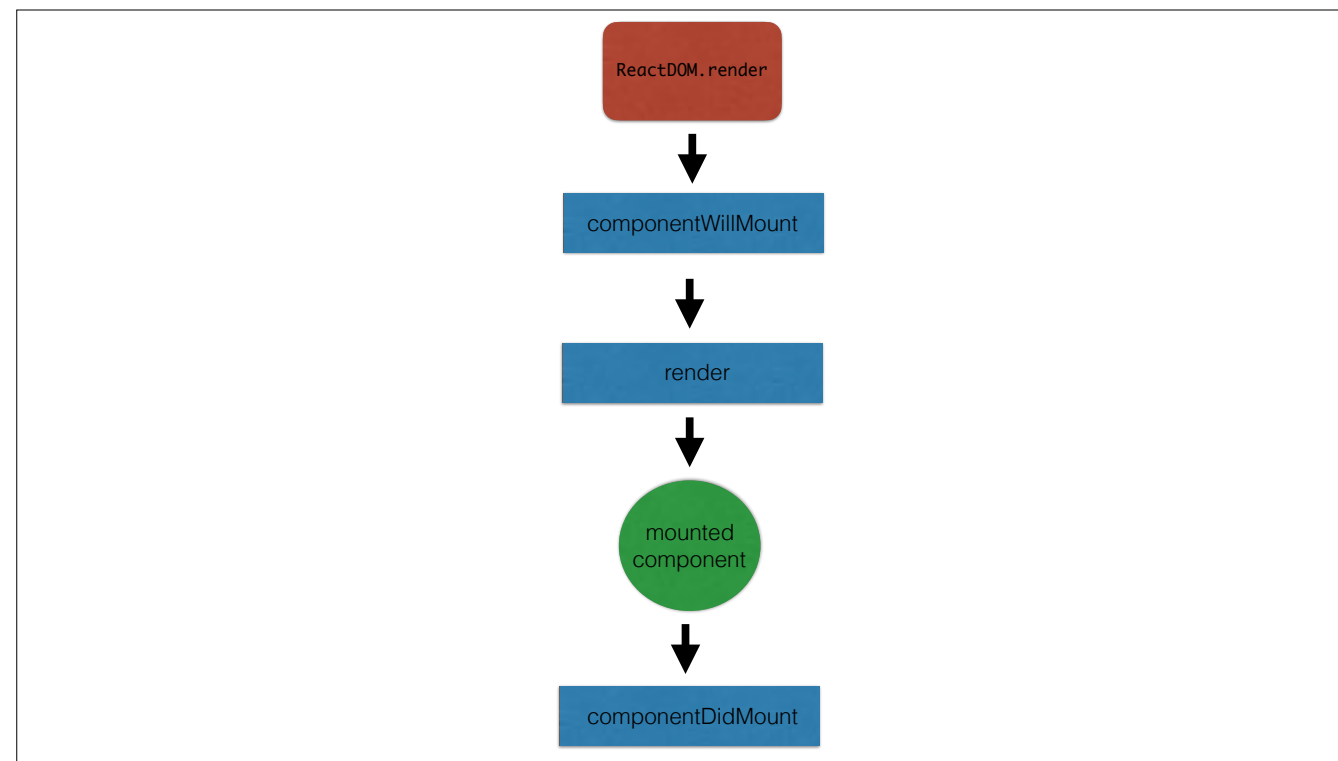
  async componentDidMount () {
    const res = await axios.get('/api/posts')
    const posts = res.data
    this.setState({posts: posts})
  }

  render () {
    // omitted for brevity
  }
}
```

Here's an example

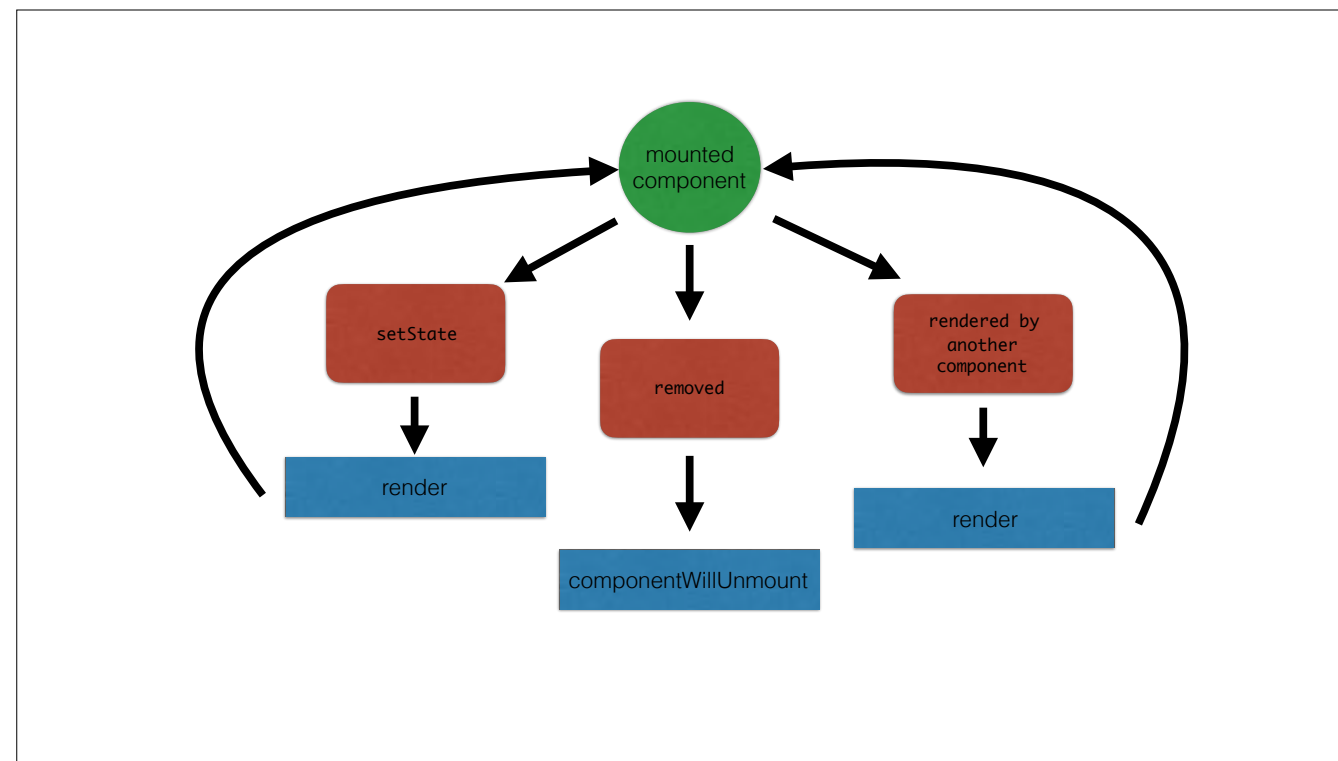
Note that this means we will be rendering initially with an empty array of posts. This is why it's important to initialize our state properly

Woah! What's axios?



Once more with feeling

SUBSEQUENT RENDERS



COMPONENT WILL UNMOUNT

- Our chance to say goodbye!
- Great place to “clean up” things
 - clear timers or intervals
 - remove event listeners

REACT LISTS

Map it out

MAP OVER LISTS

- Use `Array.prototype.map` to turn lists of data into JSX
- Great way to deal with lists and table rows
- Only stipulation: each set of JSX in the list needs to be given a special “key” prop

```
const DogList = (props) => {  
  
  return (  
    <ul>  
  
    </ul>  
  )  
}
```

```
const DogList = (props) => {  
  const puppies = props.puppies  
  
  return (  
    <ul>  
  
    </ul>  
  )  
}
```

```
const DogList = (props) => {  
  // [{id: 1, name: 'Cody'}, {id: 2, name: 'Lexie'}]  
  const puppies = props.puppies  
  
  return (  
    <ul>  
  
    </ul>  
  )  
}
```

```
const DogList = (props) => {  
  // [{id: 1, name: 'Cody'}, {id: 2, name: 'Lexie'}]  
  const puppies = props.puppies  
  
  return (  
    <ul>  
      {  
        puppies.map(puppy => <li                >{puppy.name}</li>  
      }  
    </ul>  
  )  
}
```

```
const DogList = (props) => {  
  // [{id: 1, name: 'Cody'}, {id: 2, name: 'Lexie'}]  
  const puppies = props.puppies  
  
  return (  
    <ul>  
      {  
        puppies.map(puppy => <li key={puppy.id}>{puppy.name}</li>)  
      }  
    </ul>  
  )  
}
```