Introduction to React JS

27/10/2018 FDC-Jaipur

© Vishwesh Jainkuniya

What is React.js?

- Developed by Facebook
- React is a view layer library, not a framework like Backbone, Angular etc.
- You can't use React to build a fully-functional web app

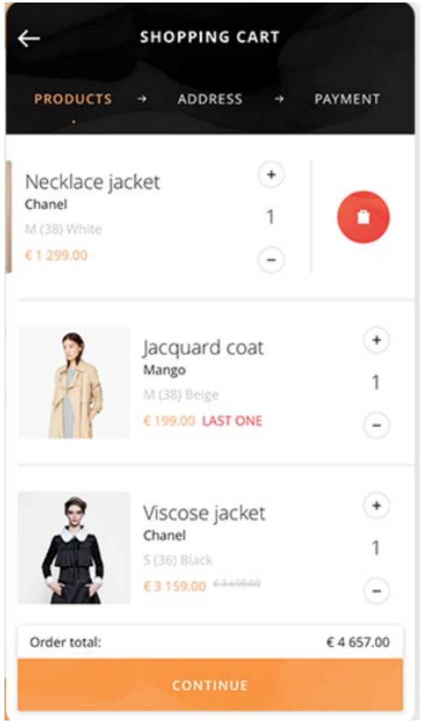
Why was React.js developed?

- Complexity of two-way data binding
- Bad UX from using "cascading updates" of DOM tree
- A lot of data on a page changing over time
- Complexity of Facebook's UI architecture

Why was React.js developed?

Binding - a strong covering holding the pages of a book together.

View


SHOPPING CART

PRODUCTS → ADDRESS → PAYMENT

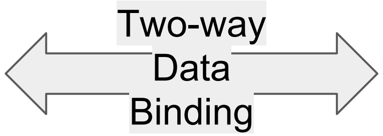
Necklace jacket
Chanel
M (38) White
€ 1 299.00
1

Jacquard coat
Mango
M (38) Beige
€ 199.00 LAST ONE
1

Viscose jacket
Chanel
S (36) Black
€ 3 159.00 ~~€ 3 499.00~~
1

Order total: € 4 657.00

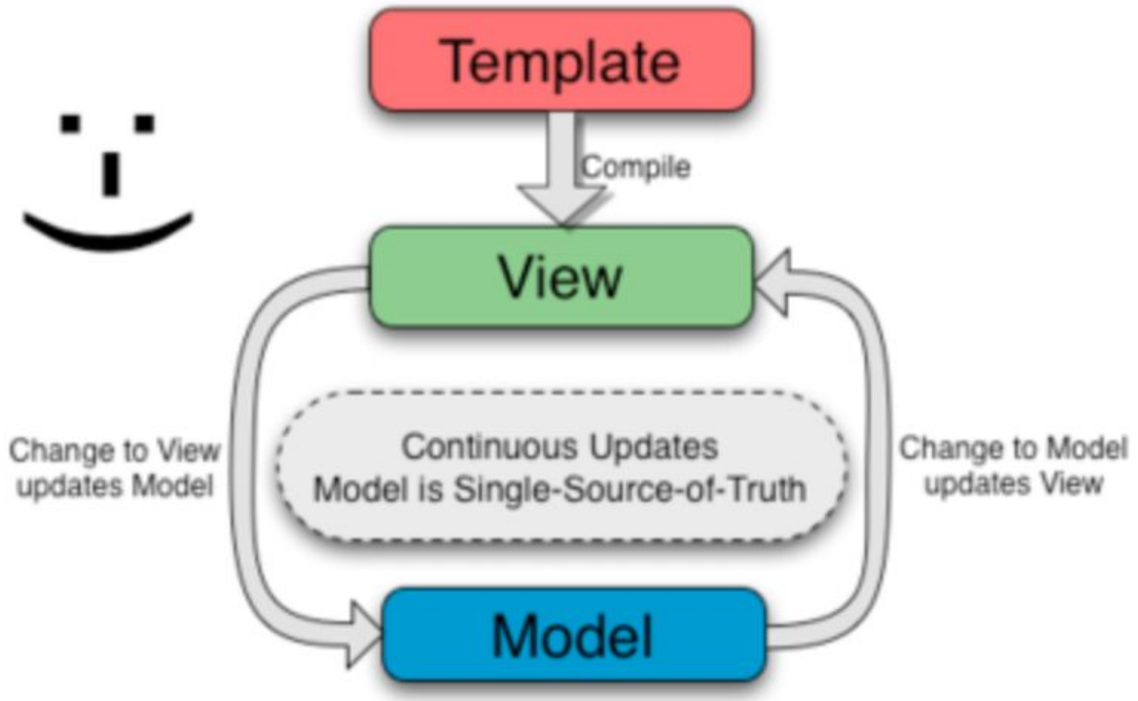
CONTINUE



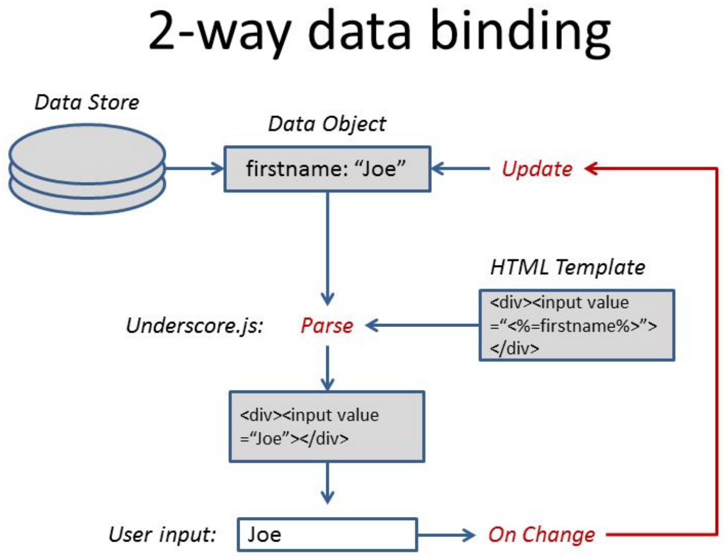
Database

```
▼ cart {3}
    1_necklace_jacket : 1
    2_jacquard_coat : 1
    3_viscose_jacket : 1
```

Two way Data Binding

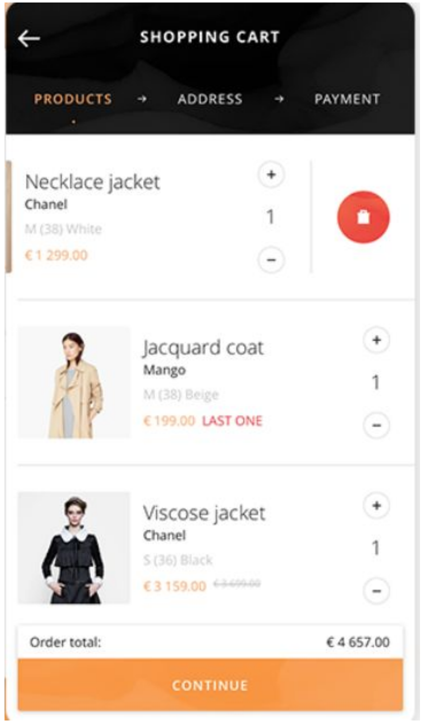

Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

Two way Data Binding



Binding

View



When user clicks on +/- button of any item:

Simple JS

- Update it's quantity in UI
- Update it's quantity in database
- Update order total

ReactJs

- Update it's quantity in State

Why was React.js developed?

- Complexity of two-way data binding
- Bad UX from using "cascading updates" of DOM tree
- A lot of data on a page changing over time
- Complexity of Facebook's UI architecture

Why was React.js developed?

- Complexity of two-way data binding
- Bad UX from using "cascading updates" of DOM tree
- A lot of data on a page changing over time
- Complexity of Facebook's UI architecture

Why was React.js developed?

- Complexity of two-way data binding
- Bad UX from using "cascading updates" of DOM tree
- A lot of data on a page changing over time
- Complexity of Facebook's UI architecture

Why was React.js developed?

- Complexity of two-way data binding
- Bad UX from using "cascading updates" of DOM tree
- A lot of data on a page changing over time
- Complexity of Facebook's UI architecture

Who uses React.js?

facebook.

TERADEK

Insta Instagram

NETFLIX

KHANACADEMY

reddit

asana

airbnb

https://github.com/facebook/react/wiki/Sites-Using-React

Why should I use React.js?

- Easy to read and understand views
- Concept of components is the future of web development
- If your page uses a lot of fast updating data or real time data - React is the way to go (eg: like chat app, real time weather app etc)
- Once react app is setup you can scale and add more feature faster

Fundamentals - React.js

- Component
- Props
- State
- JSX
- Virtual DOM

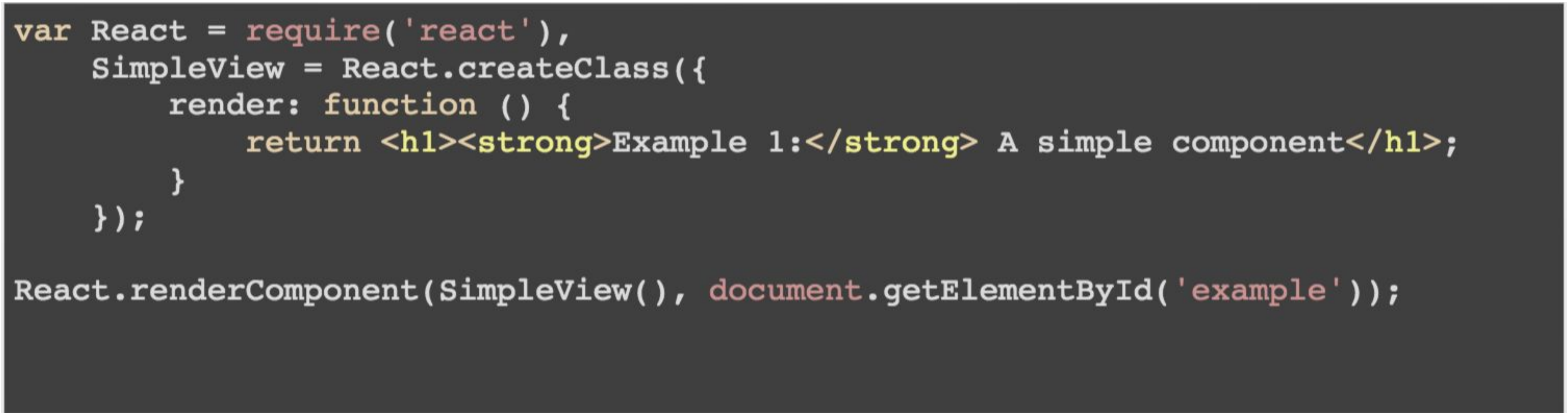
Component - React.js

- They are self-contained reusable building blocks of web application
    - Easily create list
- React components are basically just idempotent functions (pure functions)
- They describe your UI at any point in time

Component - React.js

- Created using `React.createClass()`
- The only required method is `render()`
- They describe your UI at any point in time
- Components are reusable
- Render function can contain logic
- Template can contain loops, filtering, ternary operators

Component - React.js



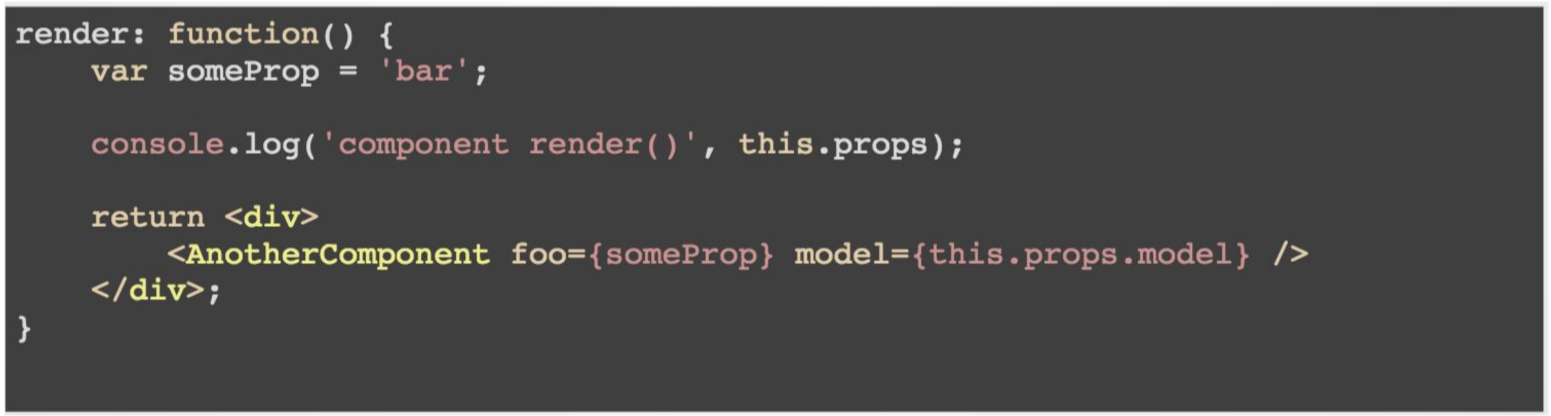
```
var React = require('react'),
    SimpleView = React.createClass({
        render: function () {
            return <h1><strong>Example 1:</strong> A simple component</h1>;
        }
    });

React.renderComponent(SimpleView(), document.getElementById('example'));
```

Component - React.js

```
var PhotoGallery = React.createClass({

  render: function() {

    var photos = this.props.photos.map(function(photo) {
      return <Photo src={photo.url} caption={photo.caption} />
    });

    return (
      <div className='photo-gallery'>
        {photos.length > 0 ? photos : ''}
      </div>
    );
  }
});
```

Props - React.js

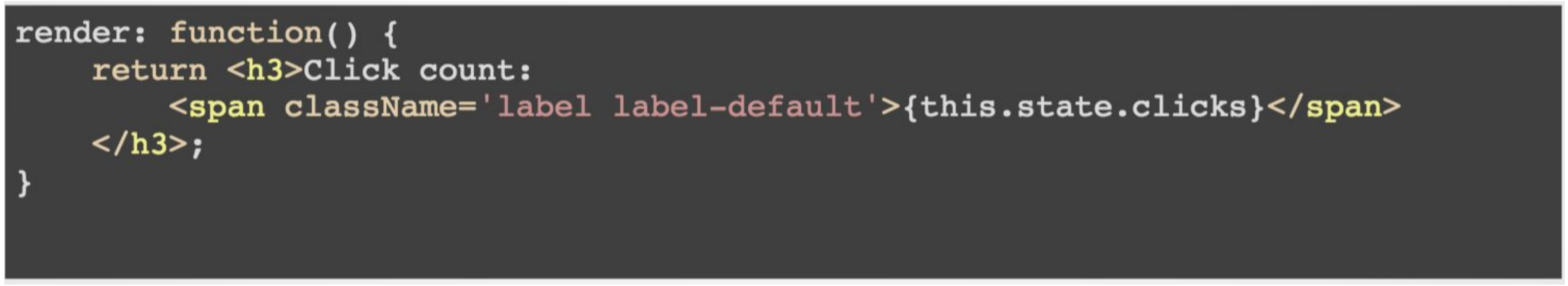
- Passed down to component from parent component and represents data for the component
- accessed via **`this.props`**



```
render: function() {
    var someProp = 'bar';

    console.log('component render()', this.props);

    return <div>
        <AnotherComponent foo={someProp} model={this.props.model} />
    </div>;
}
```

State - React.js

- Represents internal state of the component
- accessed via **`this.state`**

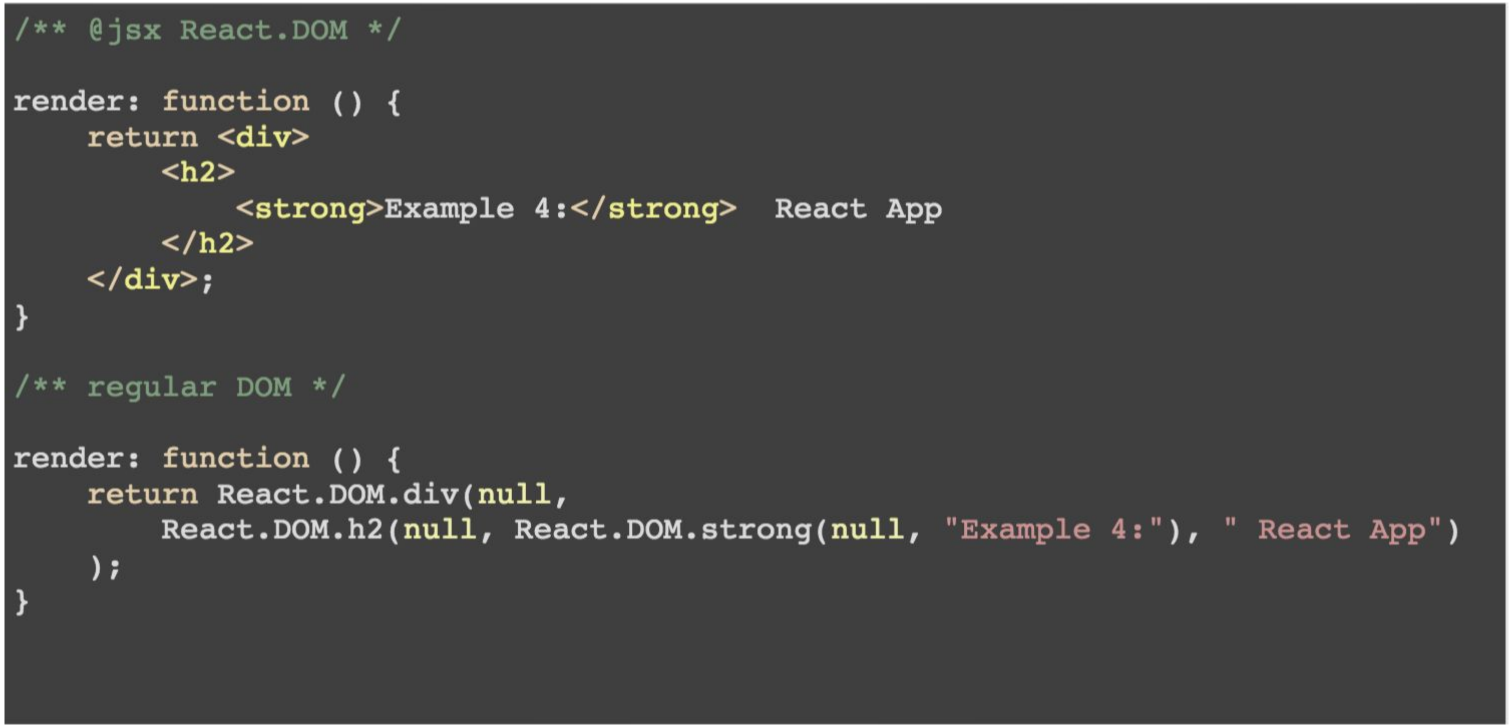


```
render: function() {
    return <h3>Click count:
        <span className='label label-default'>{this.state.clicks}</span>
    </h3>;
}
```

JSX - React.js

- one of the coolest things in React.js □
- XML-like syntax for generating component's HTML
- Easier to read and understand large DOM trees

JSX - React.js



```
/** @jsx React.DOM */

render: function () {
    return <div>
        <h2>
            <strong>Example 4:</strong>  React App
        </h2>
    </div>;
}

/** regular DOM */

render: function () {
    return React.DOM.div(null,
        React.DOM.h2(null, React.DOM.strong(null, "Example 4:"), " React App")
    );
}
```

Virtual DOM - React.js

- The virtual DOM is used for efficient re-rendering of the DOM
- React aims to re-render the virtual tree only when the state changes
- Uses 2 virtual trees (new and previous) to find differences and batch update real DOM

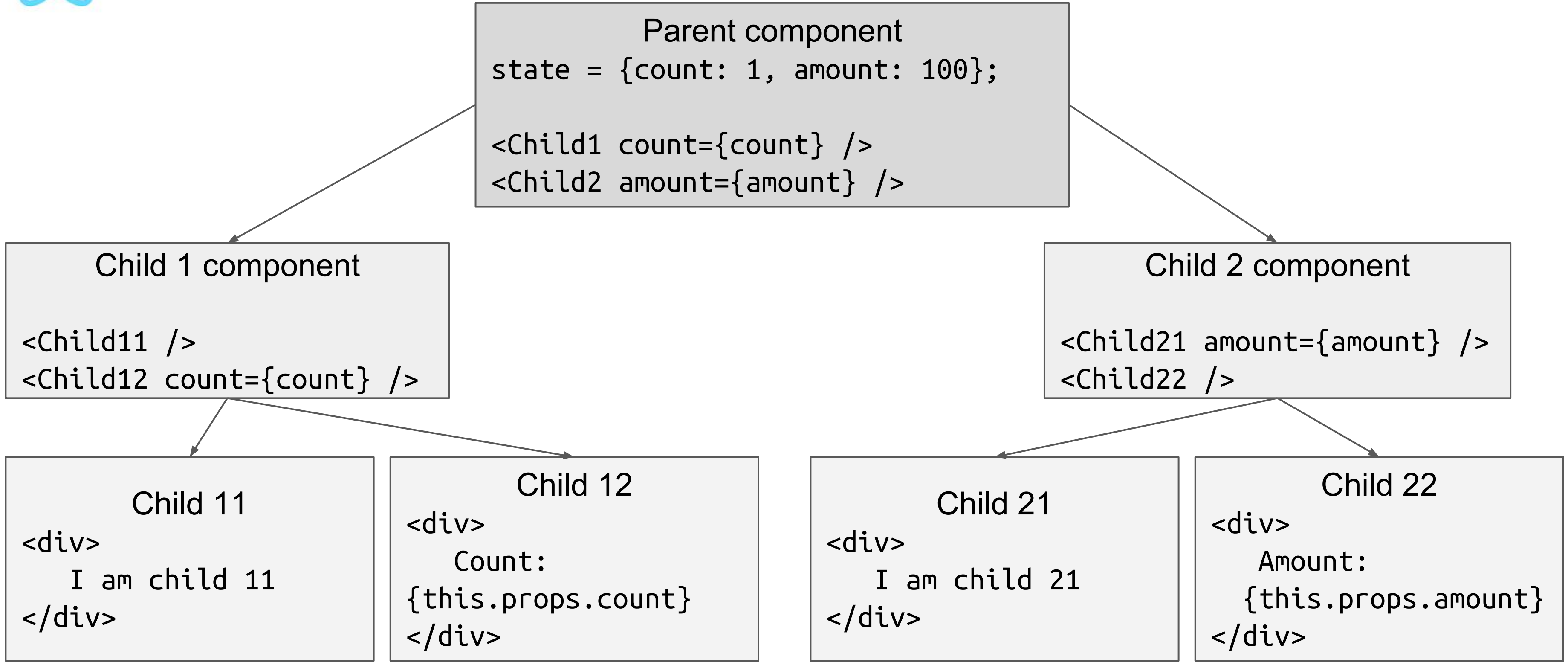
Virtual DOM - React.js

- Observes data changes (**`setState`**) and does dirty-checking to know when to re-render component
- Whenever possible, does not update entire component in real DOM - only computes a patch operation that updates part of the DOM

Component - Props - State - React.js

- When a component's **state/props** changes, the rendered markup will be updated by re-invoking **`render()`** method

Component - Props - State - React.js



Parent component
```
state = {count: 1, amount: 100};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component
```
<Child11 />
<Child12 count={count} />
```

Child 2 component
```
<Child21 amount={amount} />
<Child22 />
```

Child 11
```
<div>
   I am child 11
</div>
```

Child 12
```
<div>
   Count:
{this.props.count}
</div>
```

Child 21
```
<div>
   I am child 21
</div>
```

Child 22
```
<div>
   Amount:
  {this.props.amount}
</div>
```

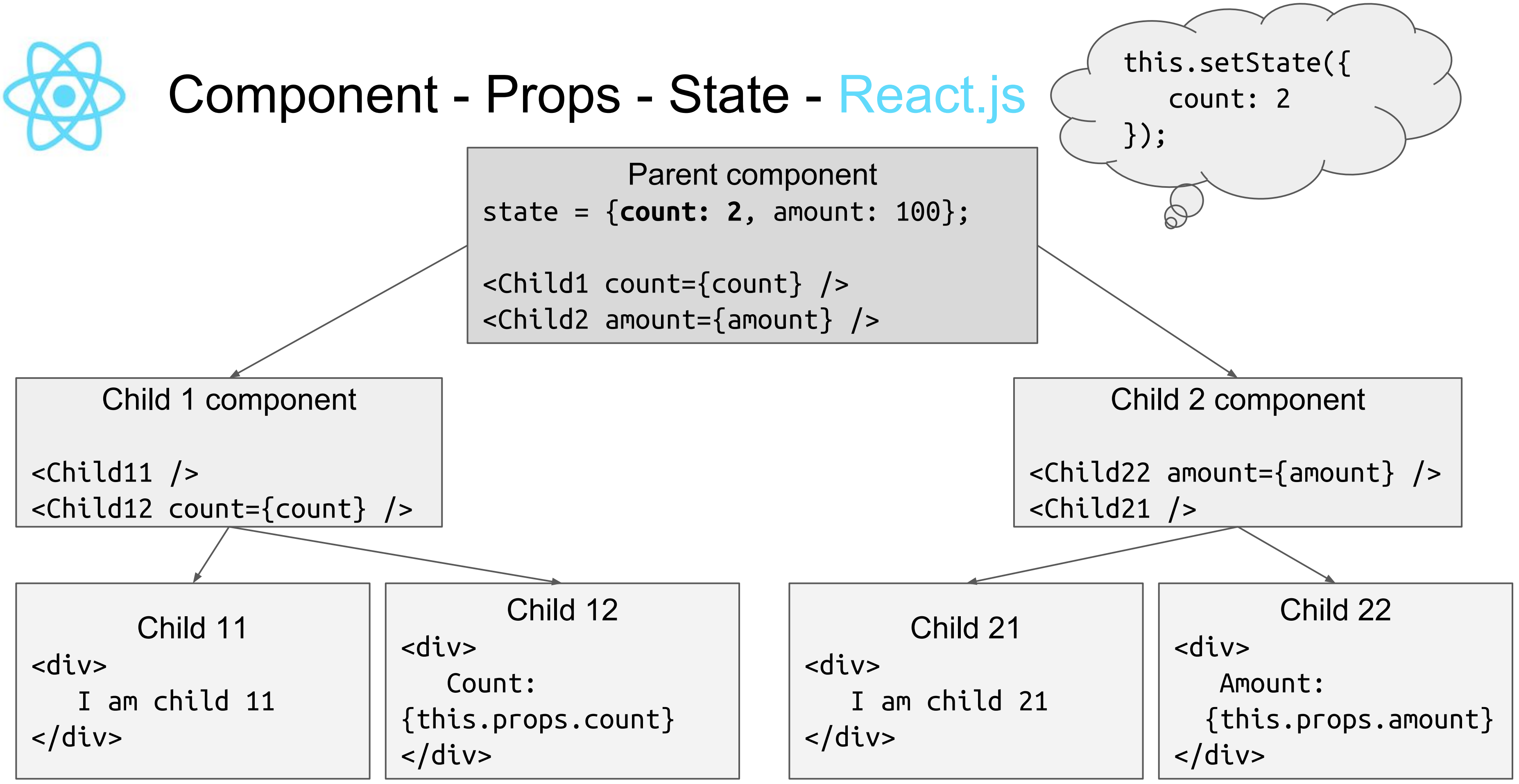
Component - Props - State - React.js

```
this.setState({
   count: 2
});
```

Parent component

```
state = {count: 1, amount: 100};

<Child1 count={count} />
<Child2 amount={amount} />
```

Component - Props - State - React.js



```
this.setState({
   count: 2
});
```

Parent component

```
state = {count: 2, amount: 100};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount} />
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

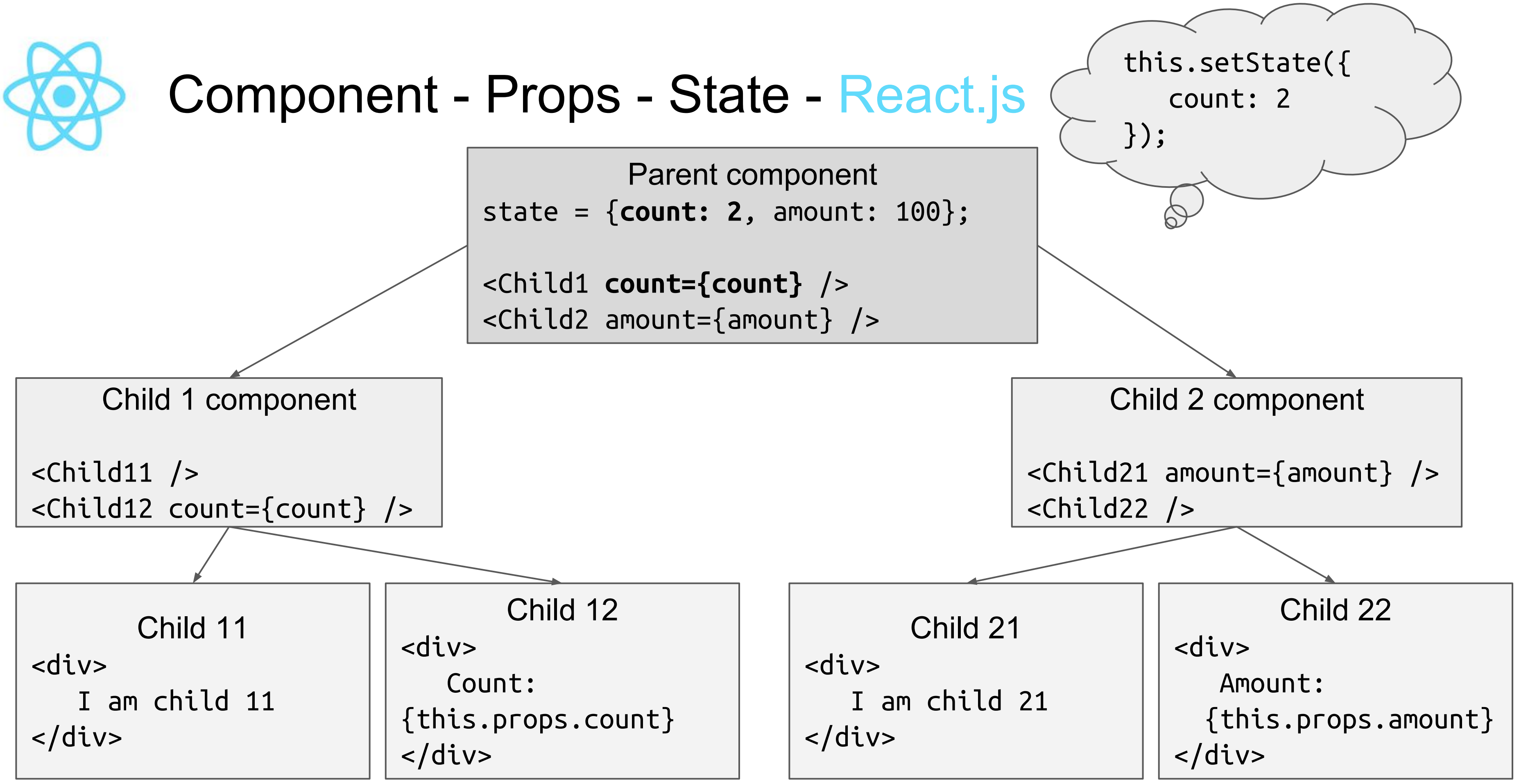
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js



```
this.setState({
   count: 2
});
```

Parent component

```
state = {count: 2, amount: 100};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child21 amount={amount} />
<Child22 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

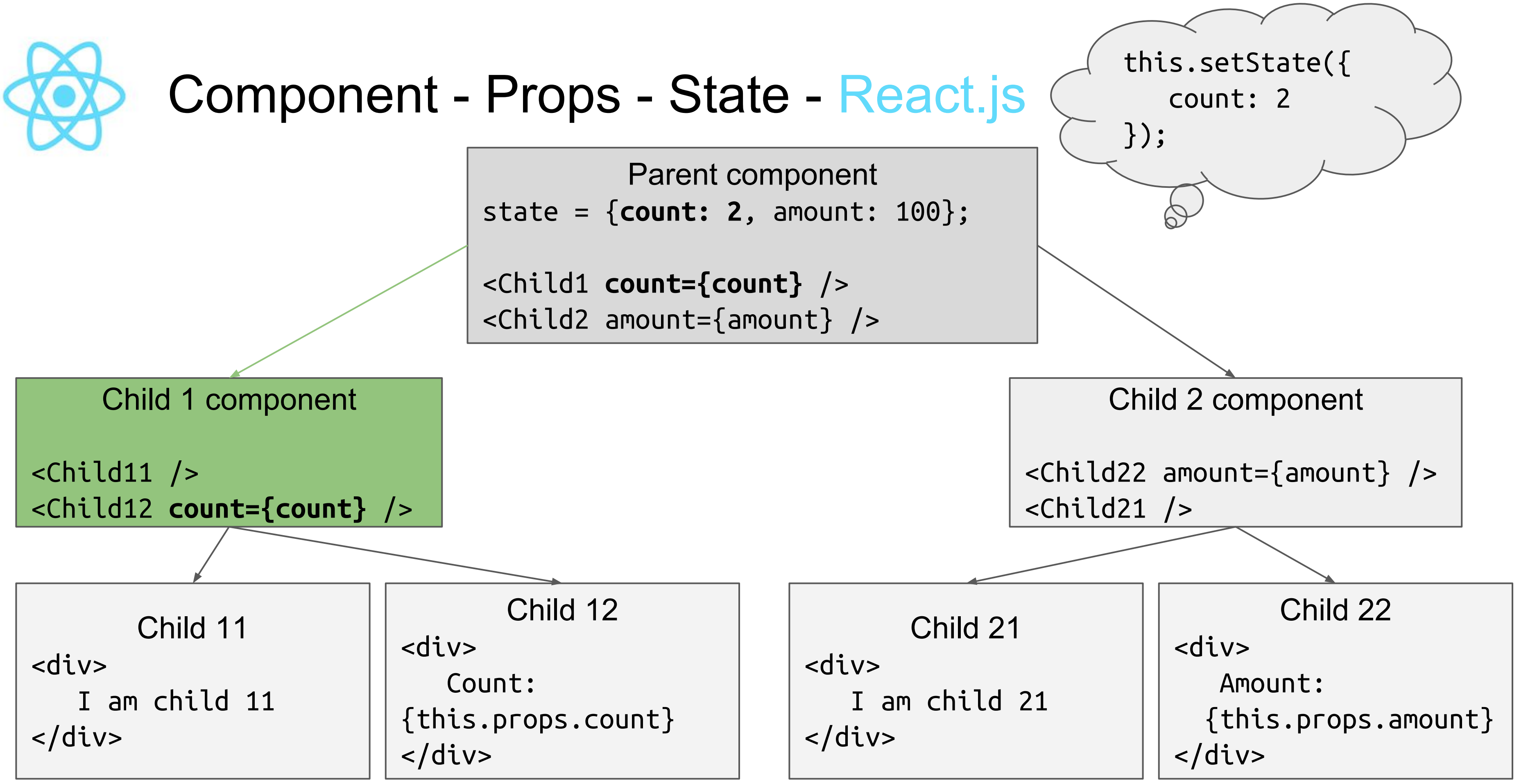
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js



```
this.setState({
    count: 2
});
```

Parent component

```
state = {count: 2, amount: 100};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount} />
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

Child 21

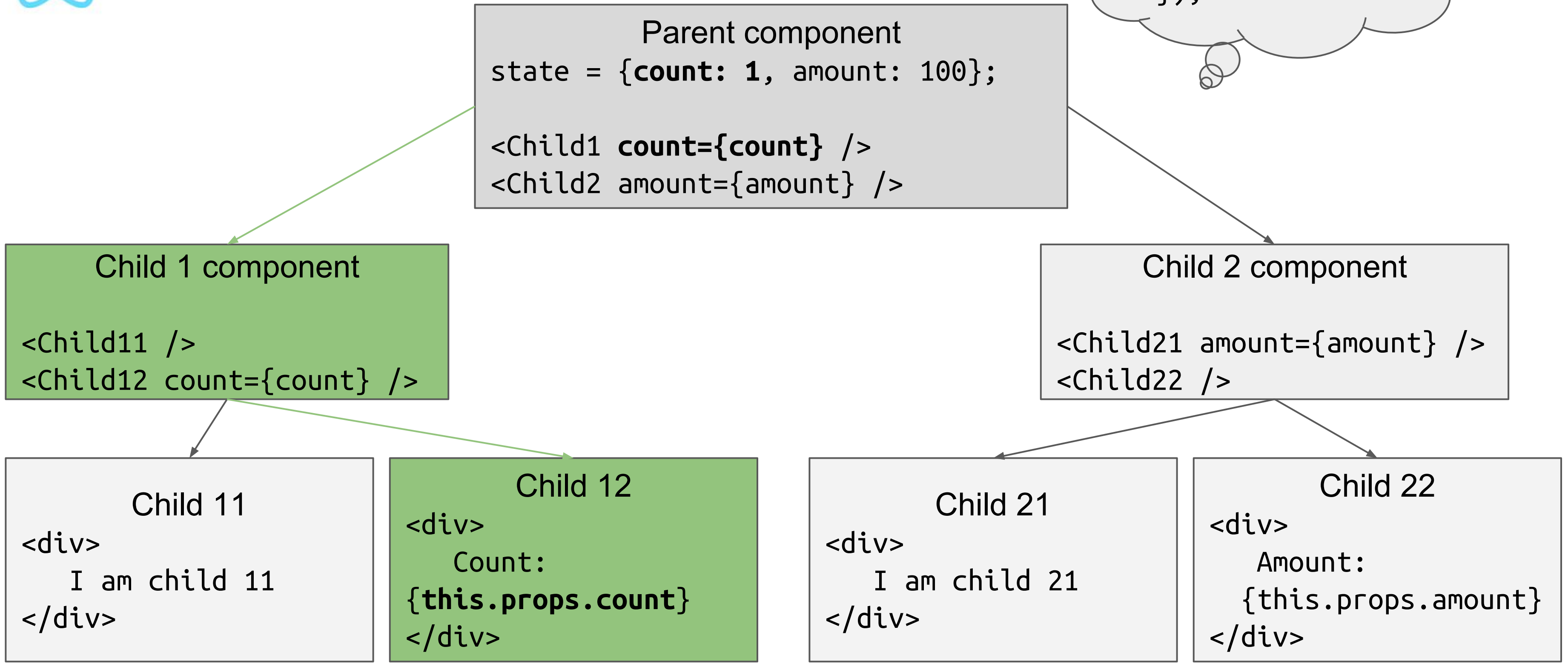
```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js

```
this.setState({
    count: 2
});
```



Parent component

```
state = {count: 1, amount: 100};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child21 amount={amount} />
<Child22 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

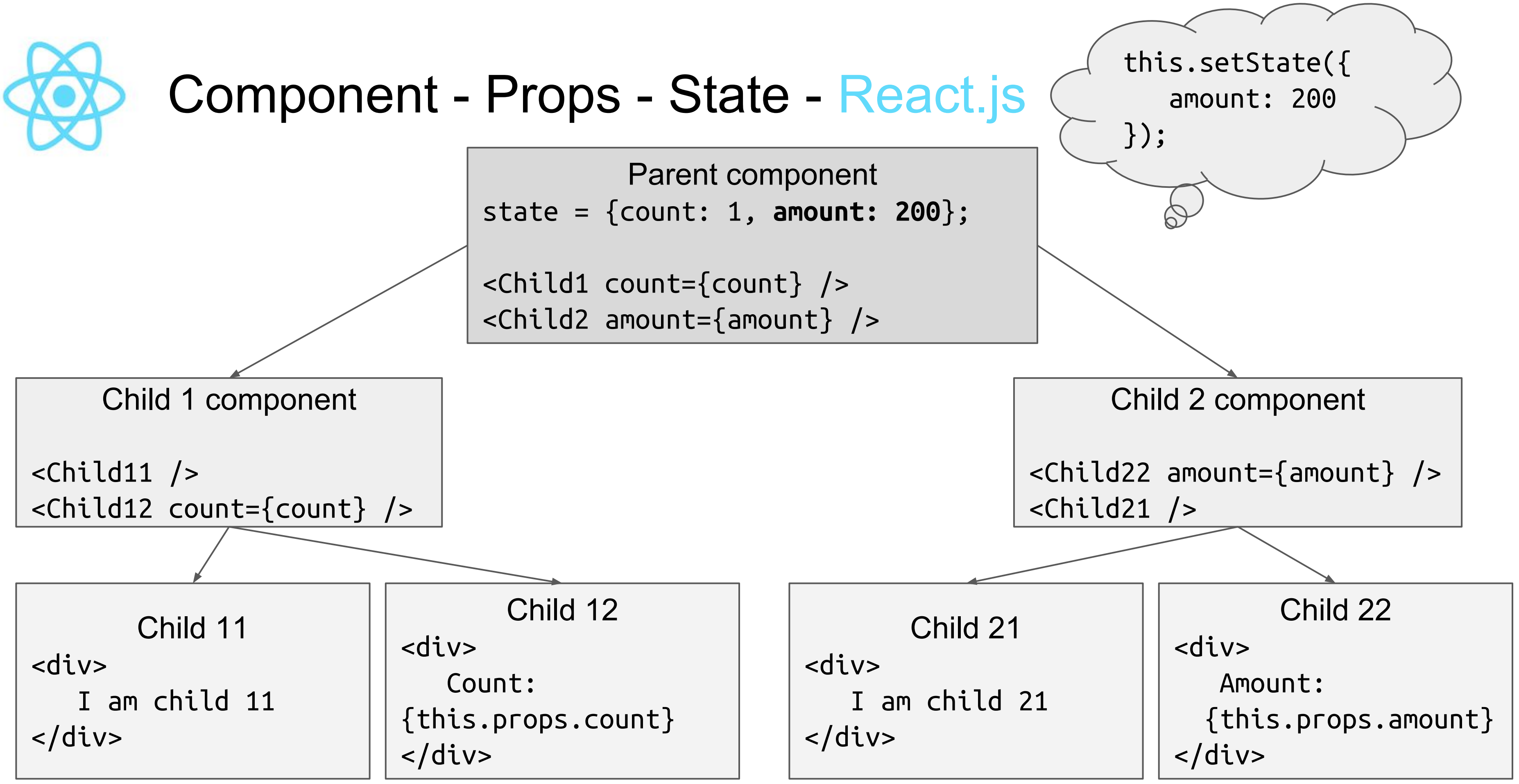
Component - Props - State - React.js

```
this.setState({
   amount: 200
});
```

Parent component

```
state = {count: 1, amount: 100};

<Child1 count={count} />
<Child2 amount={amount} />
```

Component - Props - State - React.js



```
this.setState({
   amount: 200
});
```

Parent component

```
state = {count: 1, amount: 200};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount} />
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

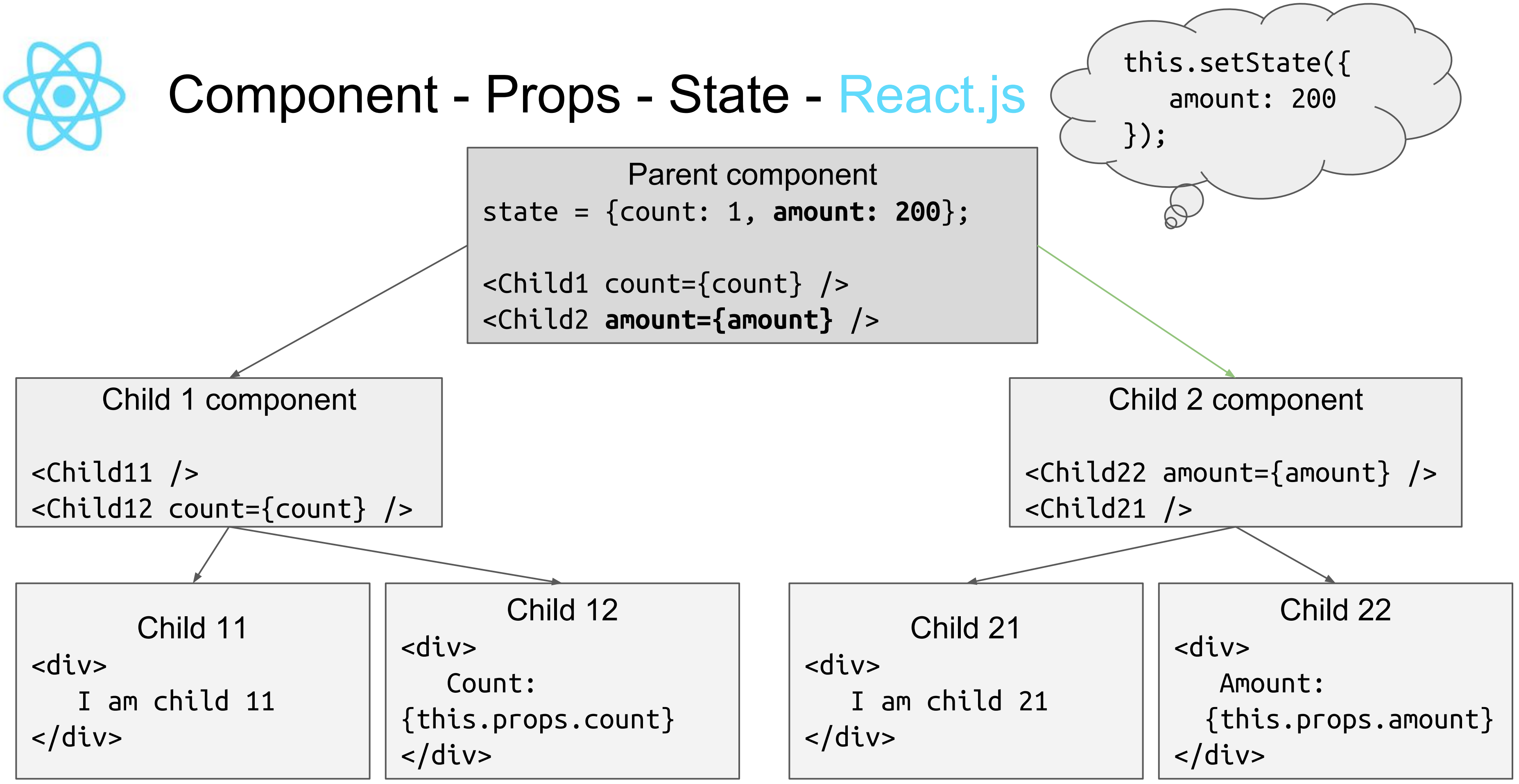
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js



```
this.setState({
   amount: 200
});
```

Parent component

```
state = {count: 1, amount: 200};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount} />
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

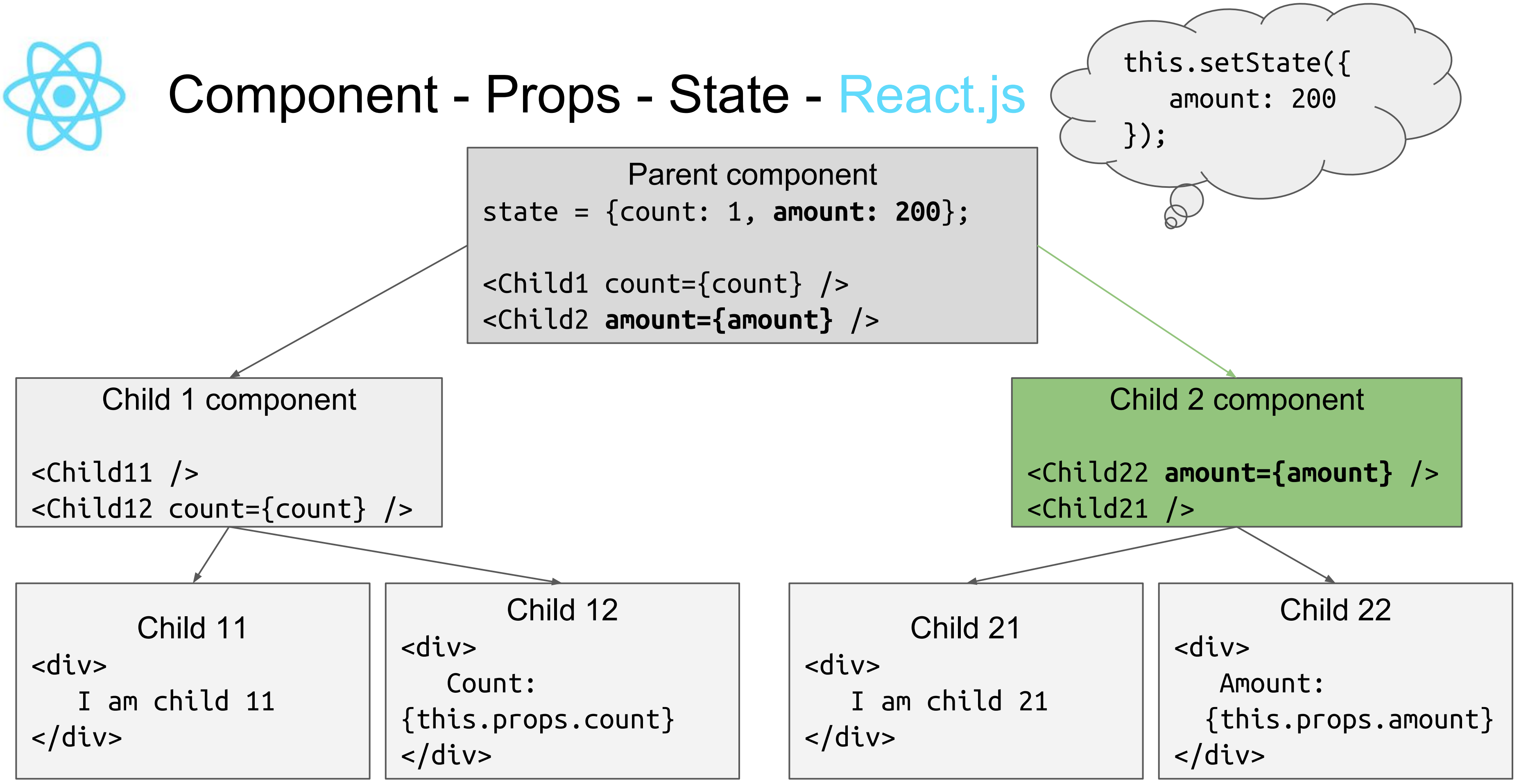
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js



```
this.setState({
    amount: 200
});
```

Parent component

```
state = {count: 1, amount: 200};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount} />
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

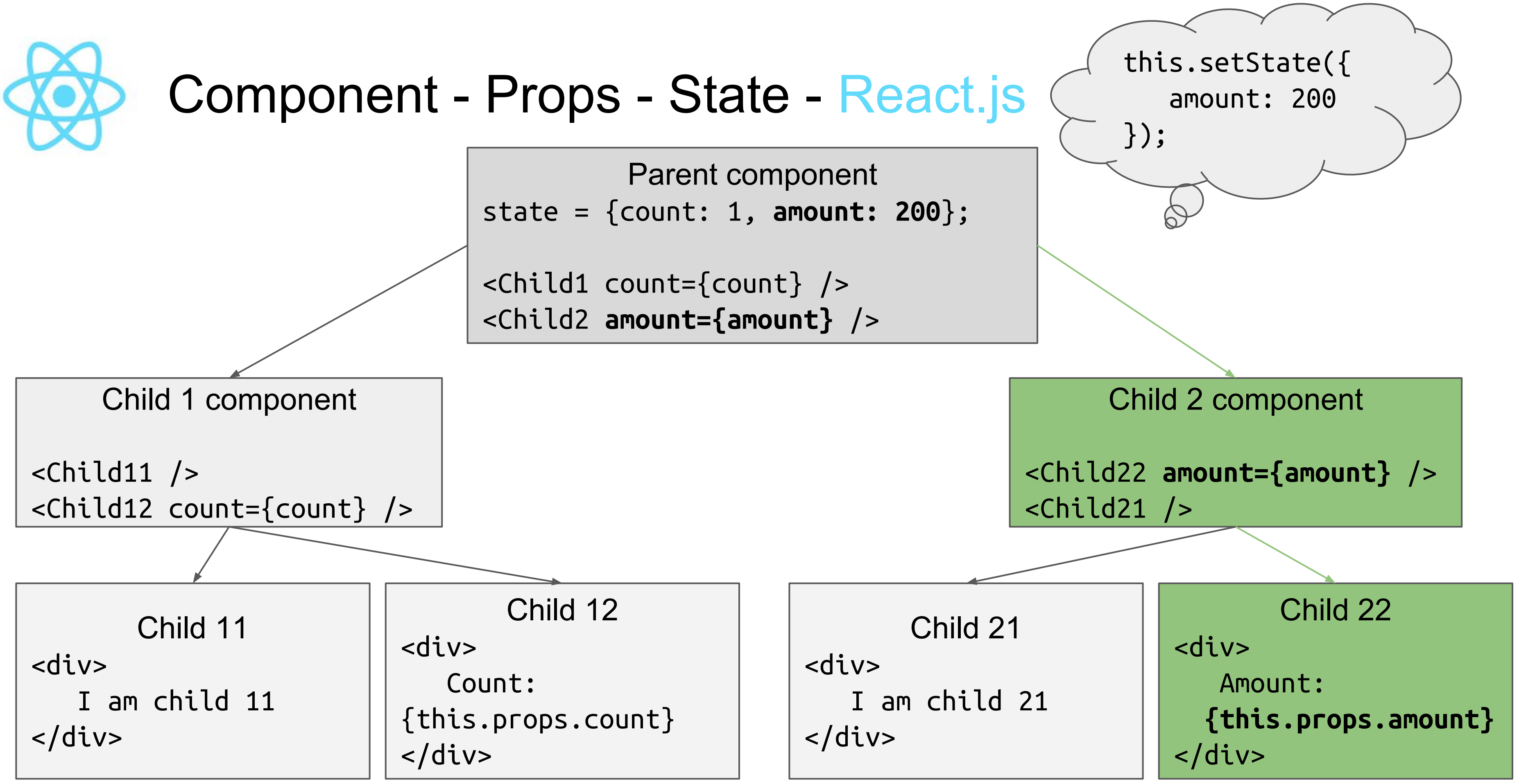
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js



```
this.setState({
    amount: 200
});
```

Parent component

```
state = {count: 1, amount: 200};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount} />
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

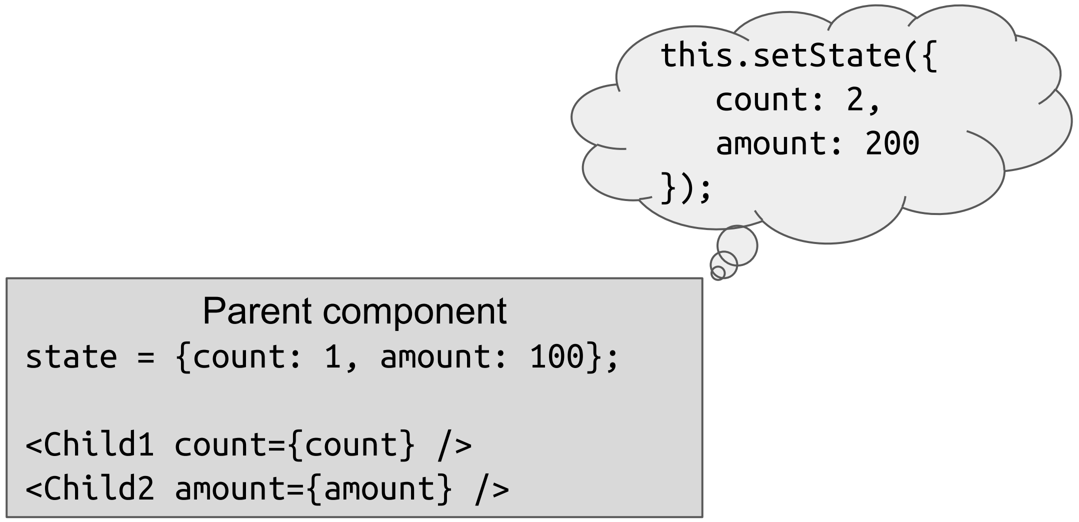
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js

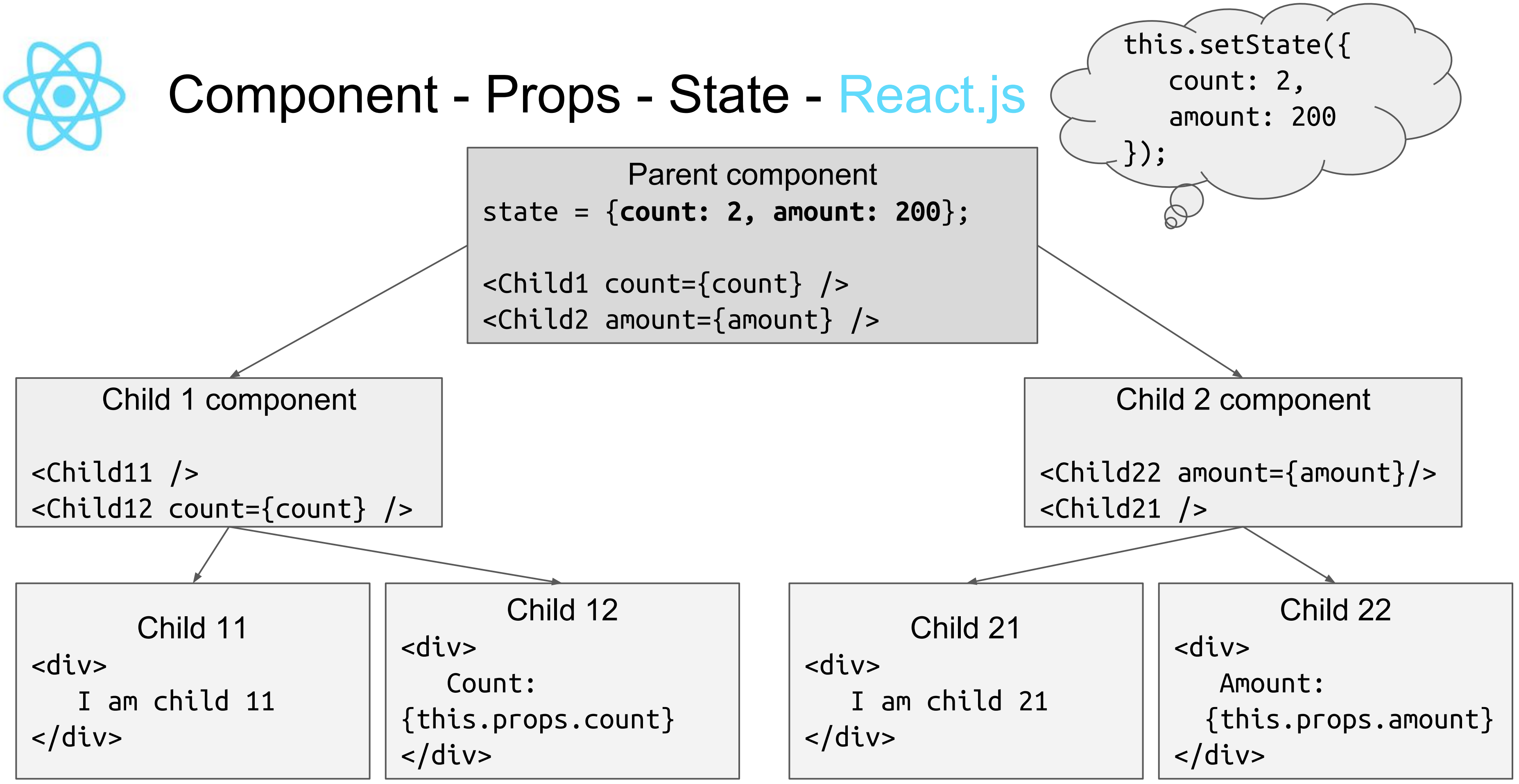


```
this.setState({
   count: 2,
   amount: 200
});
```

Parent component

```
state = {count: 1, amount: 100};

<Child1 count={count} />
<Child2 amount={amount} />
```

Component - Props - State - React.js



```
this.setState({
    count: 2,
    amount: 200
});
```

Parent component

```
state = {count: 2, amount: 200};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount}/>
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

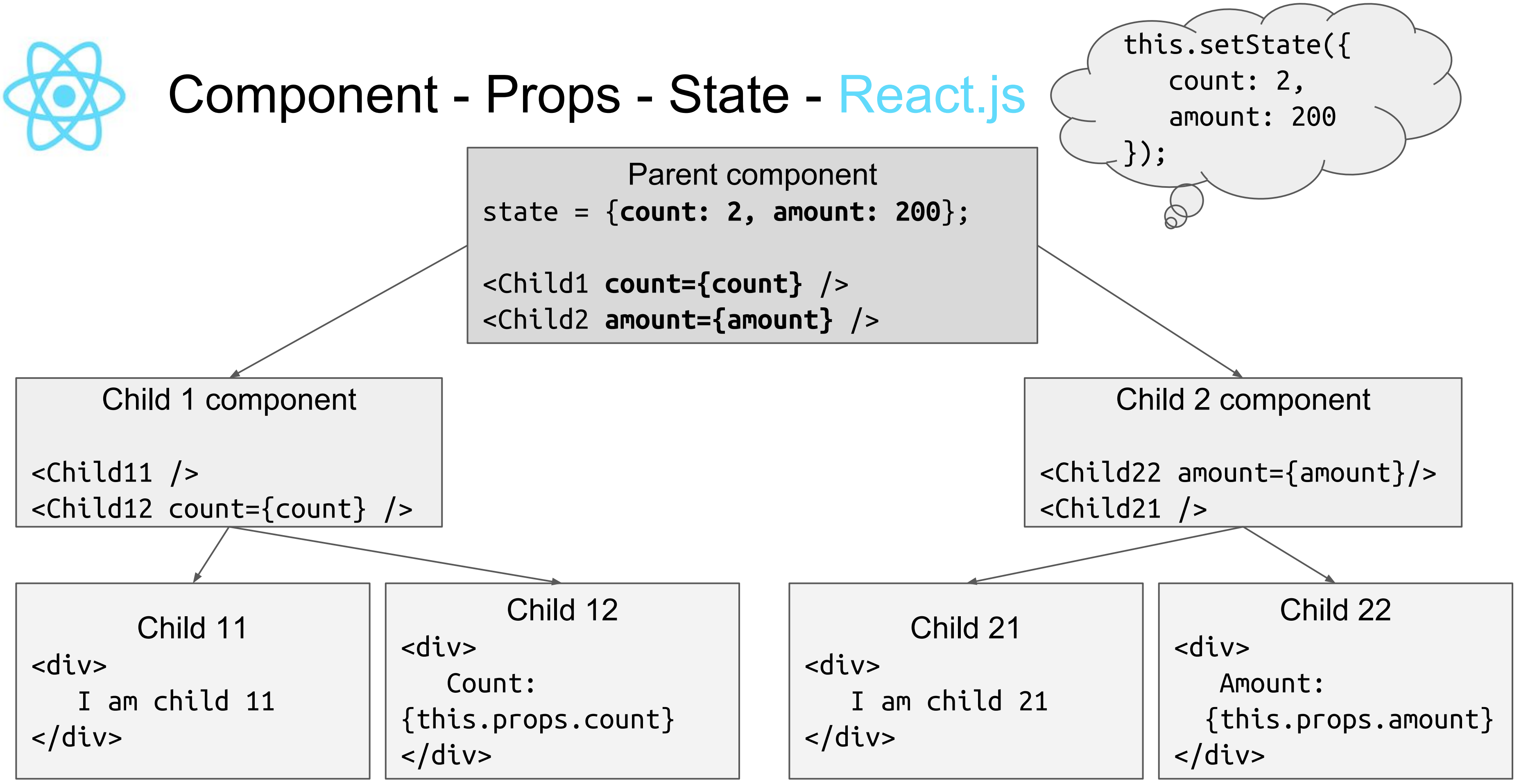
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js



```
this.setState({
   count: 2,
   amount: 200
});
```

Parent component

```
state = {count: 2, amount: 200};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount}/>
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

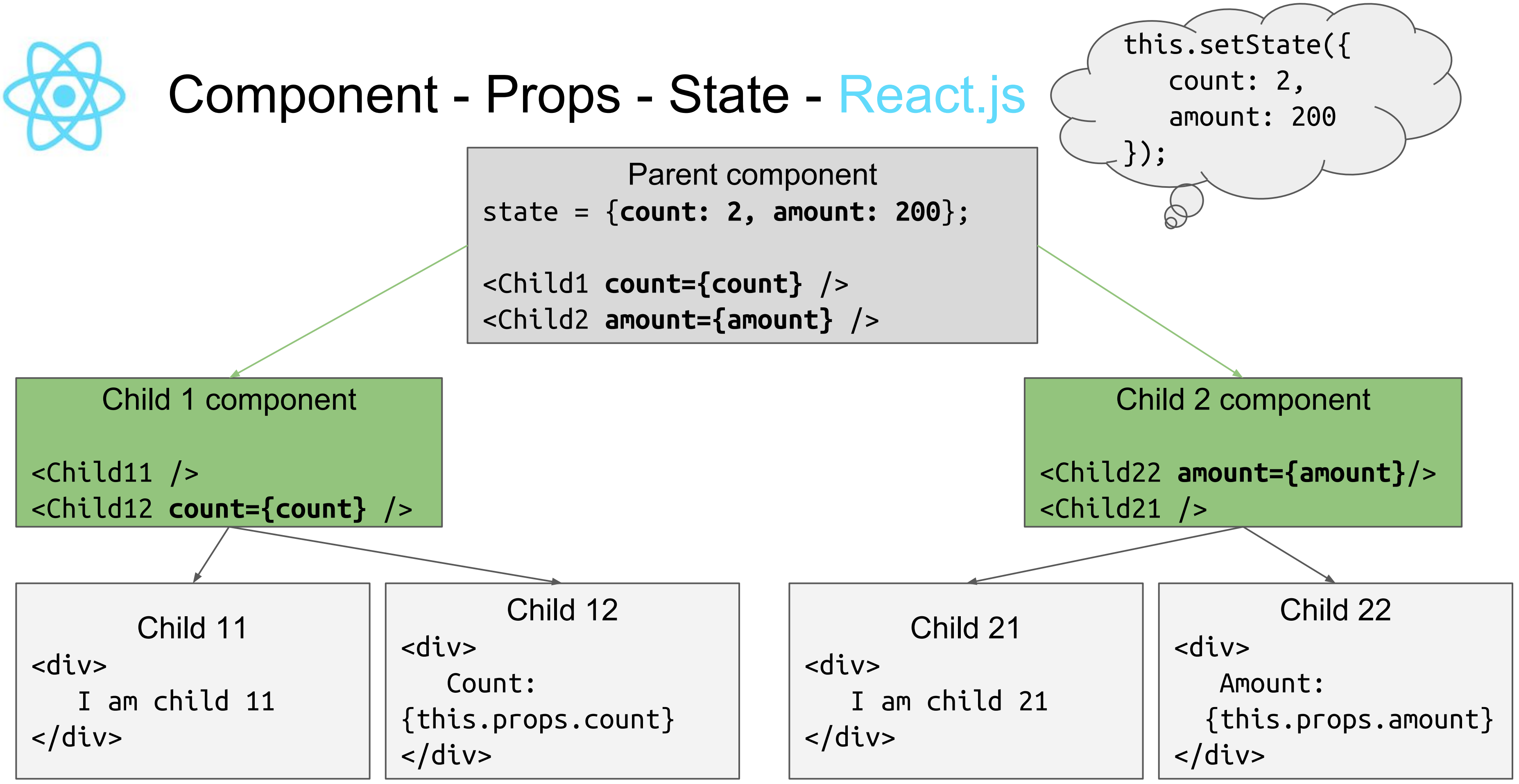
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js



```
this.setState({
    count: 2,
    amount: 200
});
```

Parent component

```
state = {count: 2, amount: 200};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount}/>
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

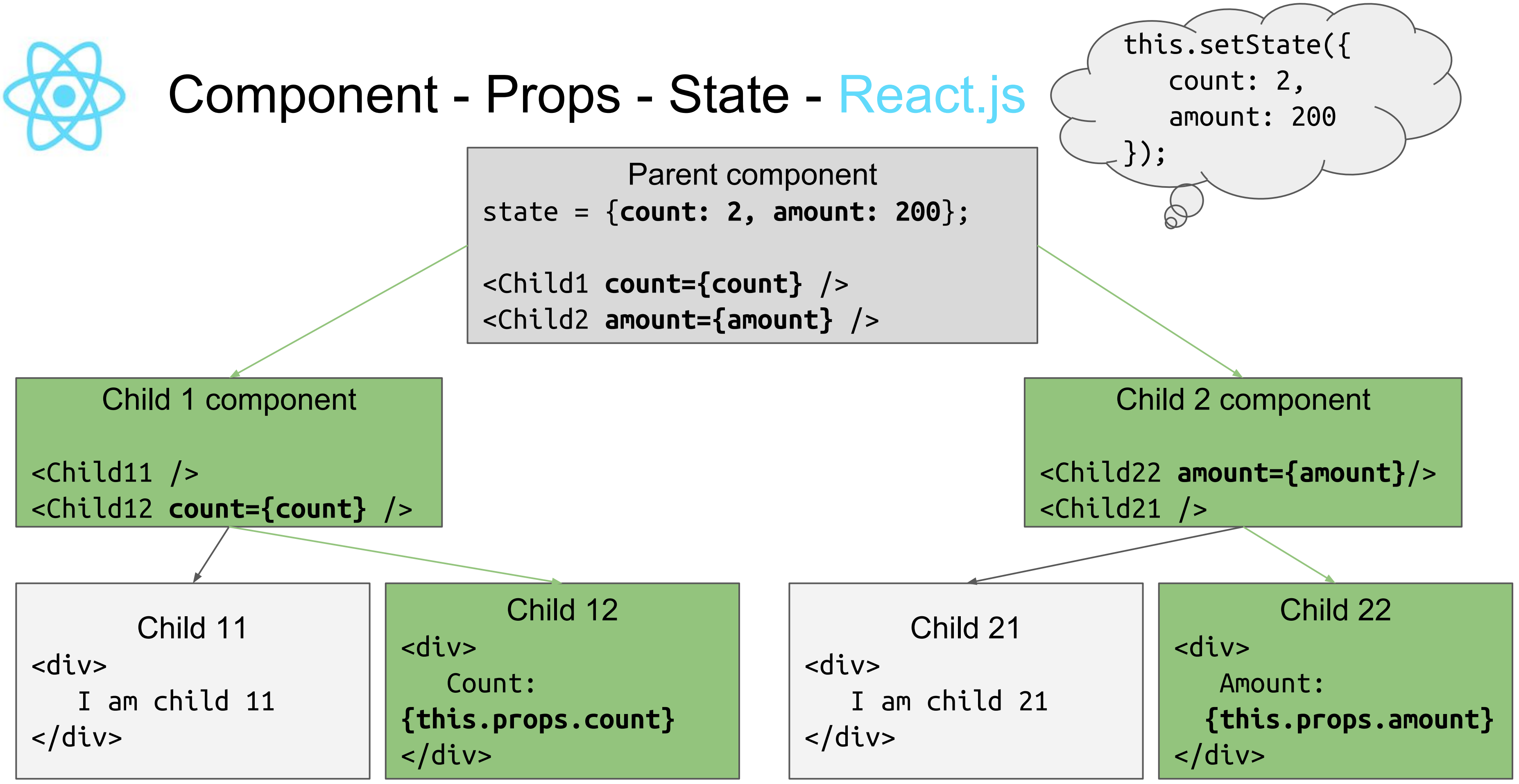
Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component - Props - State - React.js



```
this.setState({
    count: 2,
    amount: 200
});
```

Parent component

```
state = {count: 2, amount: 200};

<Child1 count={count} />
<Child2 amount={amount} />
```

Child 1 component

```
<Child11 />
<Child12 count={count} />
```

Child 2 component

```
<Child22 amount={amount}/>
<Child21 />
```

Child 11

```
<div>
   I am child 11
</div>
```

Child 12

```
<div>
   Count:
{this.props.count}
</div>
```

Child 21

```
<div>
   I am child 21
</div>
```

Child 22

```
<div>
   Amount:
  {this.props.amount}
</div>
```

Component Lifecycle - React.js

- Mounting - when an instance of a component is being created and inserted into the DOM (in order):
  - `constructor()`
  - `static getDerivedStateFromProps()`
  - `render()`
  - `componentDidMount()`

Component Lifecycle - React.js

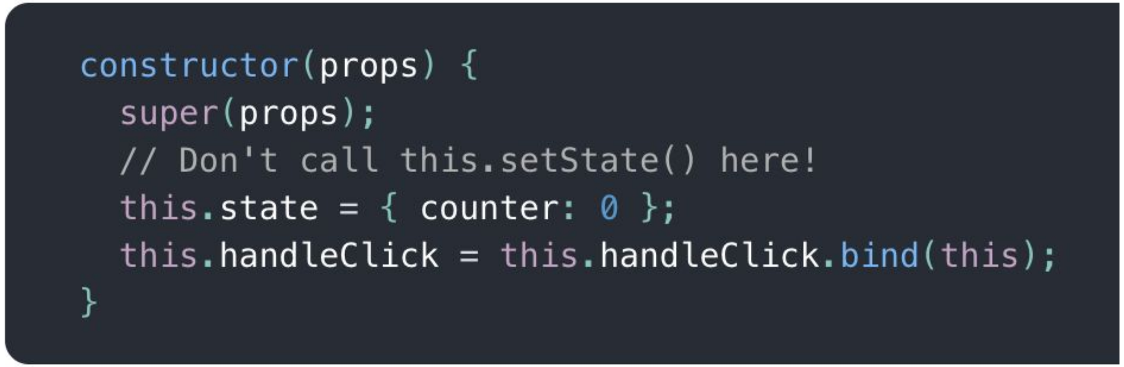
- Updating - An update can be caused by changes to props or state (in order):
  - `static getDerivedStateFromProps()`
  - `shouldComponentUpdate()`
  - `render()`
  - `getSnapshotBeforeUpdate()`
  - `componentDidUpdate()`

Component Lifecycle - React.js

- Unmounting - when a component is being removed from the DOM:
  - `componentWillUnmount()`

Component Lifecycle - React.js

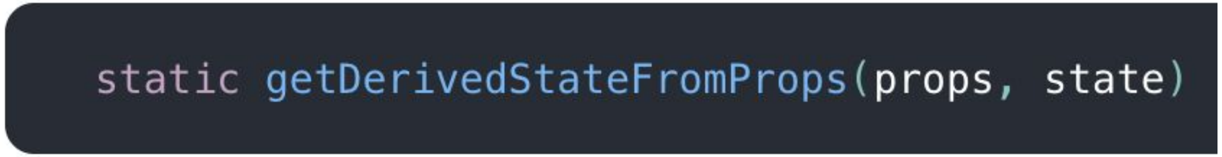
- Mounting - when an instance of a component is being created and inserted into the DOM (in order):
    - `constructor()`



```
constructor(props) {
  super(props);
  // Don't call this.setState() here!
  this.state = { counter: 0 };
  this.handleClick = this.handleClick.bind(this);
}
```

Component Lifecycle - React.js

- Mounting - when an instance of a component is being created and inserted into the DOM (in order):
  - `static getDerivedStateFromProps()`
    - is invoked right before calling the render method, both on the initial mount and on subsequent updates.
    - the state depends on changes in props over time.



```
static getDerivedStateFromProps(props, state)
```

Component Lifecycle - React.js

- Mounting - when an instance of a component is being created and inserted into the DOM (in order):
  - `render()`
    - UI of the component, React elements, Booleans or null etc
    - should be pure, meaning that it does not modify component state, it returns the same result each time it's invoked, and it does not directly interact with the browser.
    - Do not call **`this.setState({});`**

Component Lifecycle - React.js

- Mounting - when an instance of a component is being created and inserted into the DOM (in order):
  - `componentDidMount()`
    - invoked immediately after a component is mounted (inserted into the tree).
    - load data from a remote endpoint
    - set up any subscriptions, unsubscribe in **`componentWillUnmount()`**.
    - you may call **`this.setState({})`** immediately in **`componentDidMount()`**. It will trigger an extra rendering

Component Lifecycle - React.js

- Updating - An update can be caused by changes to props or state (in order):
    - **`static getDerivedStateFromProps()`**
    - `shouldComponentUpdate()`
    - `render()`
    - `getSnapshotBeforeUpdate()`
    - `componentDidUpdate()`

Component Lifecycle - React.js

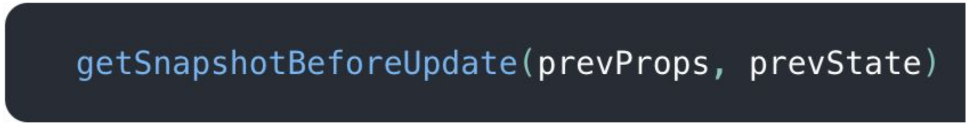
- Updating - An update can be caused by changes to props or state (in order):
  - `shouldComponentUpdate()`
    - is invoked before rendering when new props or state are being received. Defaults to true. This method is not called for the initial render or when `forceUpdate()` is used.
    - Automatically implemented in **`PureComponent`.**
    - Avoid calling **`this.setState({})`**

Component Lifecycle - React.js

- Updating - An update can be caused by changes to props or state (in order):
  - **`static getDerivedStateFromProps()`**
  - **`shouldComponentUpdate()`**
  - **`render()`**
  - `getSnapshotBeforeUpdate()`
  - `componentDidUpdate()`

Component Lifecycle - React.js

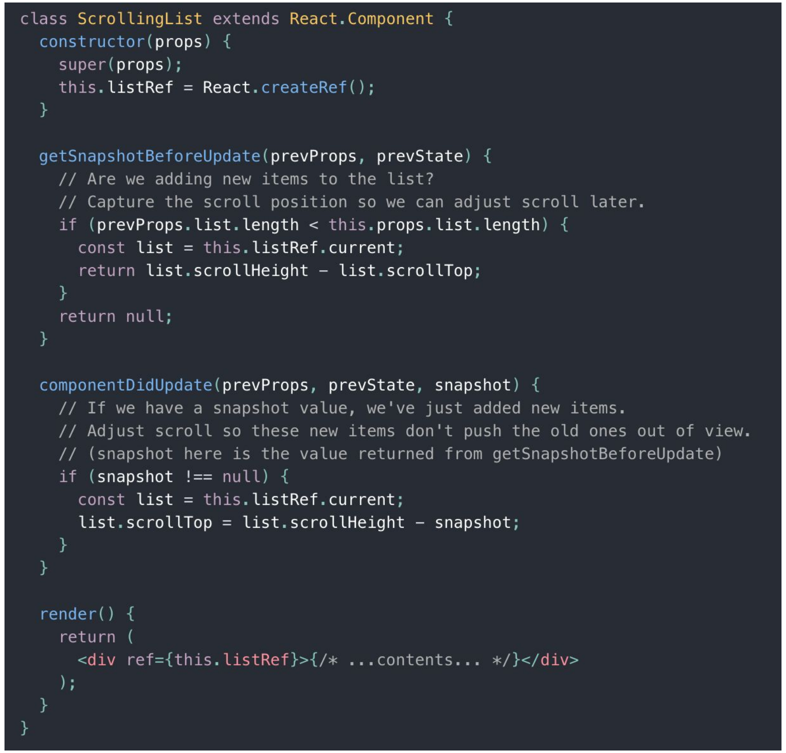
- Updating - An update can be caused by changes to props or state (in order):
  - `getSnapshotBeforeUpdate()`
    - invoked right before the most recently rendered output is committed to e.g. the DOM. It enables your component to capture some information from the DOM (e.g. scroll position) before it is potentially changed. Any value returned by this lifecycle will be passed `as a parameter to` **`componentDidUpdate()`**.



```
getSnapshotBeforeUpdate(prevProps, prevState)
```

Component Lifecycle - React.js

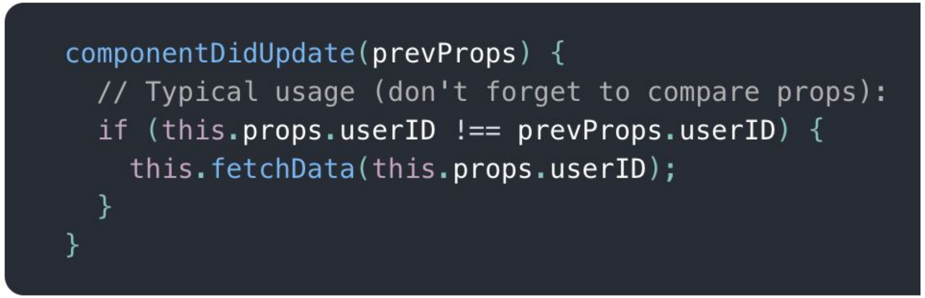
- Updating - An update can be caused by changes to props or state (in order):
    - `getSnapshotBeforeUpdate()`



```
class ScrollingList extends React.Component {
  constructor(props) {
    super(props);
    this.listRef = React.createRef();
  }

  getSnapshotBeforeUpdate(prevProps, prevState) {
    // Are we adding new items to the list?
    // Capture the scroll position so we can adjust scroll later.
    if (prevProps.list.length < this.props.list.length) {
      const list = this.listRef.current;
      return list.scrollHeight - list.scrollTop;
    }
    return null;
  }

  componentDidUpdate(prevProps, prevState, snapshot) {
    // If we have a snapshot value, we've just added new items.
    // Adjust scroll so these new items don't push the old ones out of view.
    // (snapshot here is the value returned from getSnapshotBeforeUpdate)
    if (snapshot !== null) {
      const list = this.listRef.current;
      list.scrollTop = list.scrollHeight - snapshot;
    }
  }

  render() {
    return (
      <div ref={this.listRef}>{/* ...contents... */}</div>
    );
  }
}
```

Component Lifecycle - React.js

- Updating - An update can be caused by changes to props or state (in order):
    - `componentDidUpdate()`
        - is invoked immediately after updating occurs. This method is not called for the initial render.
        - do network requests as long as you compare the current props to previous props.



```
componentDidUpdate(prevProps) {
  // Typical usage (don't forget to compare props):
  if (this.props.userID !== prevProps.userID) {
    this.fetchData(this.props.userID);
  }
}
```

Component Lifecycle - React.js

- Unmounting - when a component is being removed from the DOM:
  - `componentWillUnmount()`
    - is invoked immediately before a component is unmounted and destroyed.
    - invalidating timers, canceling network requests, or cleaning up any subscriptions

Component Lifecycle - React.js

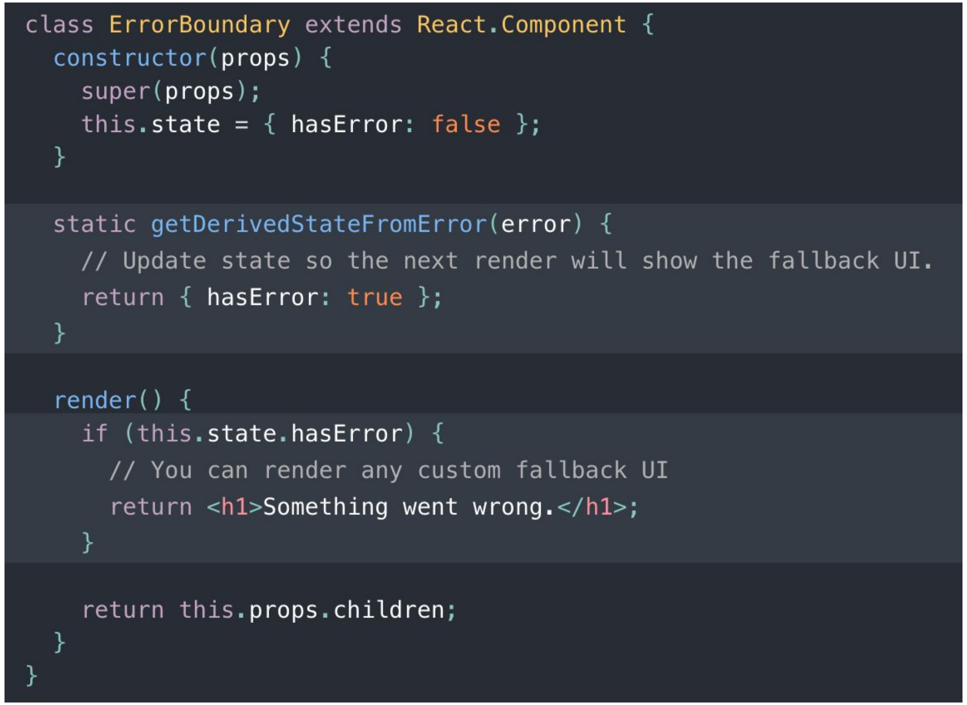
- Error Handling - is an error during rendering, in a lifecycle method, or in the constructor of any child component:
  - `static getDerivedStateFromError()`
  - `componentDidCatch()`

Component Lifecycle - React.js

- Error Handling - is an error during rendering, in a lifecycle method, or in the constructor of any child component:
  - `static getDerivedStateFromError()`
    - is invoked after an error has been thrown by a descendant component. It receives the error that was thrown as a parameter and should return a value to update state.

Component Lifecycle - React.js

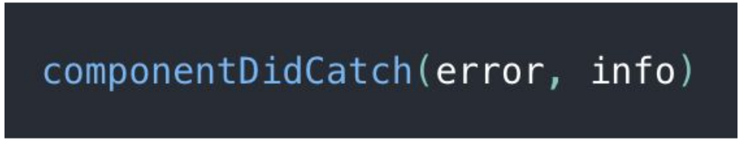
- Error Handling - is an error during rendering, in a lifecycle method, or in the constructor of any child component:
  - `static getDerivedStateFromError()`



```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

Component Lifecycle - React.js

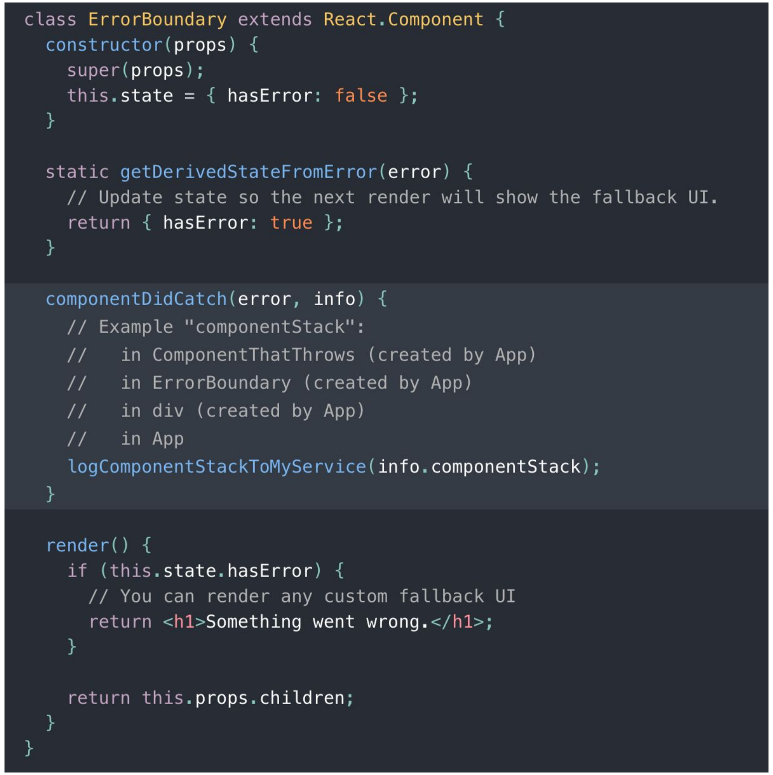
- Error Handling - is an error during rendering, in a lifecycle method, or in the constructor of any child component:
    - `componentDidCatch()`
        - invoked after an error has been thrown by a descendant component. It receives two parameters:
            - error - The error that was thrown.
            - info - An object with a componentStack key containing information about which component threw the error.



```
componentDidCatch(error, info)
```

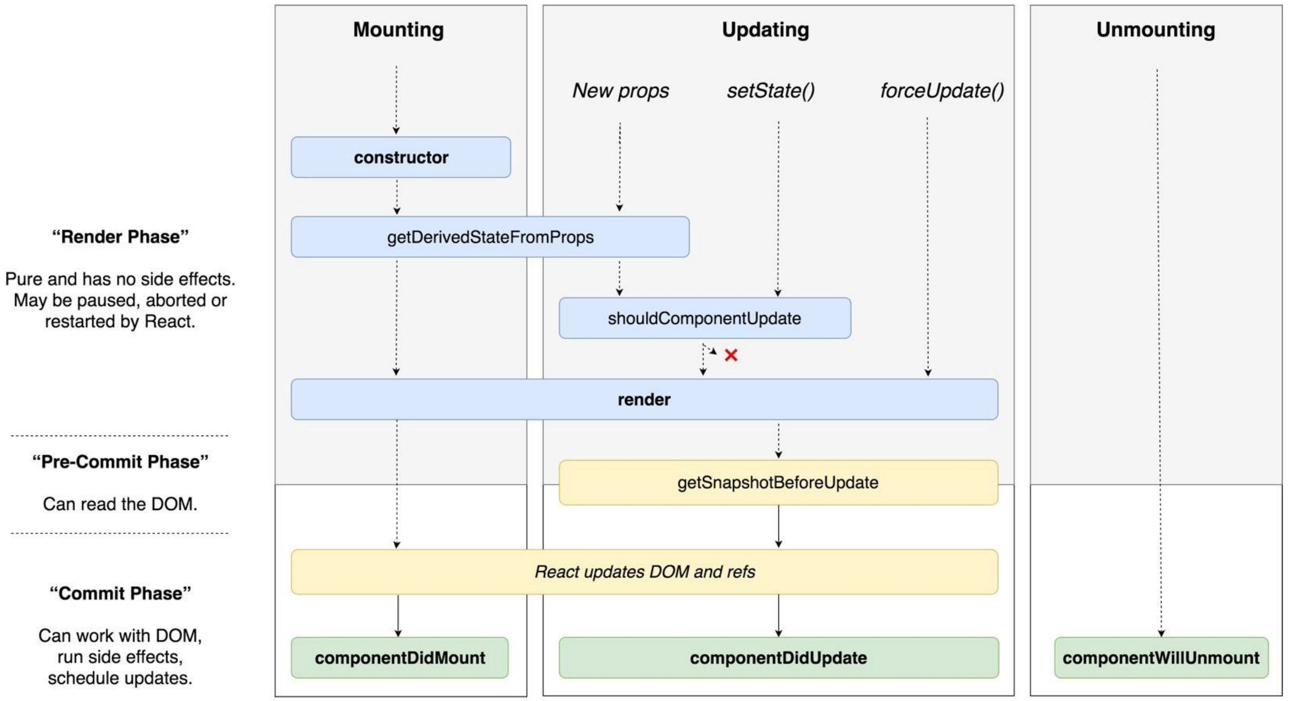
Component Lifecycle - React.js

- Error Handling - is an error during rendering, in a lifecycle method, or in the constructor of any child component:
    - `componentDidCatch()`



```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  componentDidCatch(error, info) {
    // Example "componentStack":
    //   in ComponentThatThrows (created by App)
    //   in ErrorBoundary (created by App)
    //   in div (created by App)
    //   in App
    logComponentStackToMyService(info.componentStack);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

Component Lifecycle - React.js



Examples 1 - React.js

https://codepen.io/bradleyboy/pen/OPBpGw

Examples 2 - React.js

https://codesandbox.io/embed/p9lz63rw8x?codemirror=1

Examples 3 - React.js

https://codepen.io/ReactJSTraining/pen/jyYjeW

Examples 4 - React.js

https://codepen.io/marekdano/pen/bVNYpq

Examples 5 - React.js

Tic Tac Toe - A ReactJS Tutorial

https://codepen.io/cdtinney/pen/OWzaeJ

Examples 6 - React.js

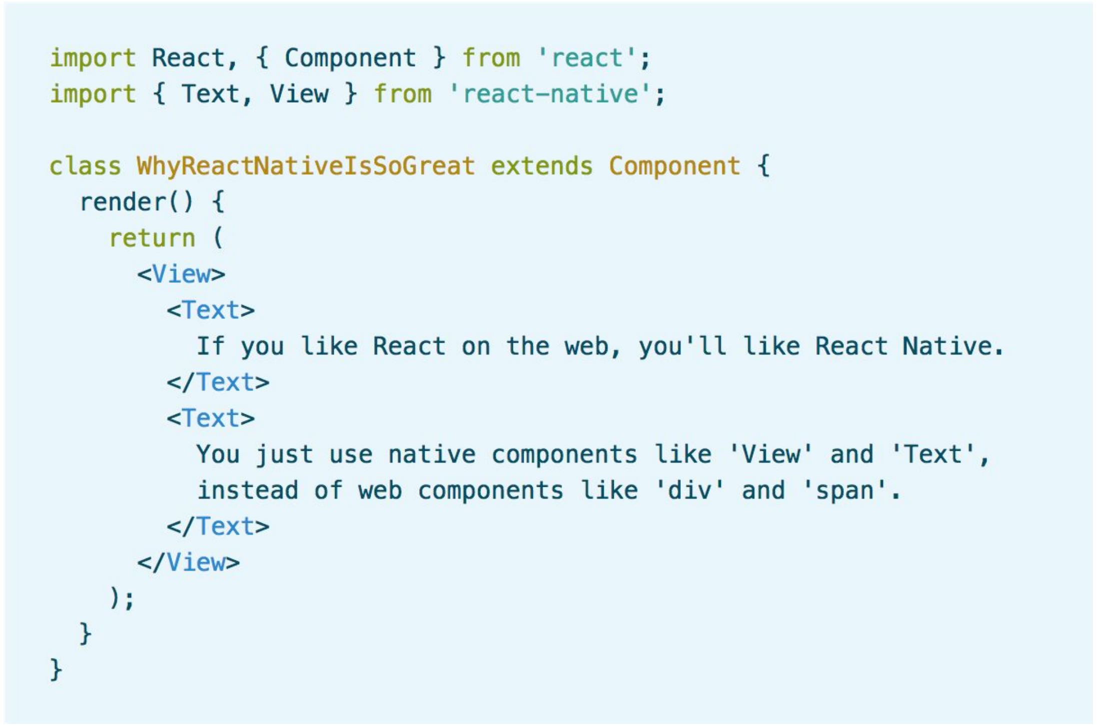
https://brandonstilson.com/dodgygame/

https://github.com/bbstilson/react-dodgy-game

React Native - React.js

- React Native is framework for building mobile apps on native platforms using JavaScript + React
- Application logic is written and runs in JavaScript, application UI is fully native
- Learn once, write anywhere - iOS, Android, Web
- Uses CSS properties for styling

React Native - React.js



```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

class WhyReactNativeIsSoGreat extends Component {
  render() {
    return (
      <View>
        <Text>
          If you like React on the web, you'll like React Native.
        </Text>
        <Text>
          You just use native components like 'View' and 'Text',
          instead of web components like 'div' and 'span'.
        </Text>
      </View>
    );
  }
}
```

Thanks You - React.js

Vishwesh Jainkuniya

Software Developer @ Zulip | Software Developer Intern @ Goibibo | GSoC '17 @ Zulip | Learner | Explorer | Full stack Developer | RN Dev | Open Source Enthusiastic | Programmer

@jainkuniya   /in/jainkuniya/   /jainkuniya

References

- http://slides.com/alexanderfarennikov/react-js-fundamentals
- https://www.linkedin.com/pulse/react-js-introduction-presentation-sergii-sema/
- https://reactjs.org/docs/react-component.html