# Schedule

**Recall:**

- Dynamic data – attribute key
- Fetching data from API

**Today:**

- Component life cycle
- React events

- React routes
- React forms (optional)

# Display collection data in JSX

IMPORTANT: **key** attribute?

```
class App extends React.Component {
  constructor() {
  super();
  this.state = [
    {
    name: 'CongNV',
    email: 'congnv@hanu.edu.vn'
    },
    {
    name: 'CamNH',
    email: 'camnh@hanu.edu.vn'
    }
  ];
  }
```

```
render() {
  return <div class="card-list">
  {
    this.state.monsters.map(monster => {
    return <div
      className='card-container’
      key={ monster.email }>

      <img src='' alt= '' />
      <h2> name </h2>
      <p> email </p>
    </div>;
    })
  }
  </div>;
}
```

## Fetching data from API

**[example: monsters]**

```
class App extends React.Component {
  constructor() {
    super();

    this.state = {
      monsters : [ ]
    };
  }

  async componentDidMount() {
    const response =
                await fetch('https://jsonplaceholder.typicode.com/users');
    const users = await response.json();

    this.setState({ monsters : users });
  }
```

componentDidMount() ?

render() ?

## Component Lifecycle

## React component lifecycle

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

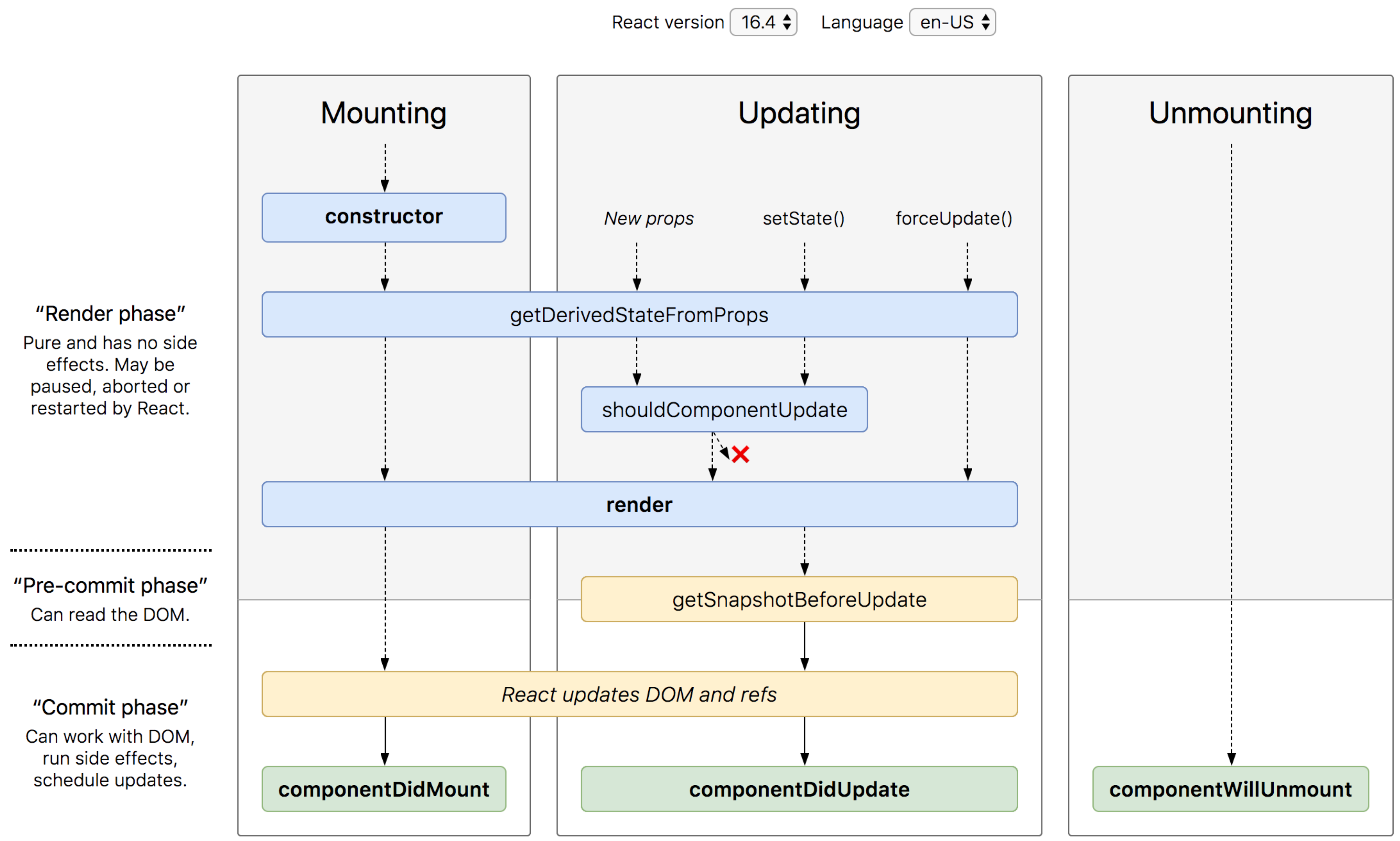
The three phases are:

- **Mounting**,
- **Updating**,
- and **Unmounting**.

Read:

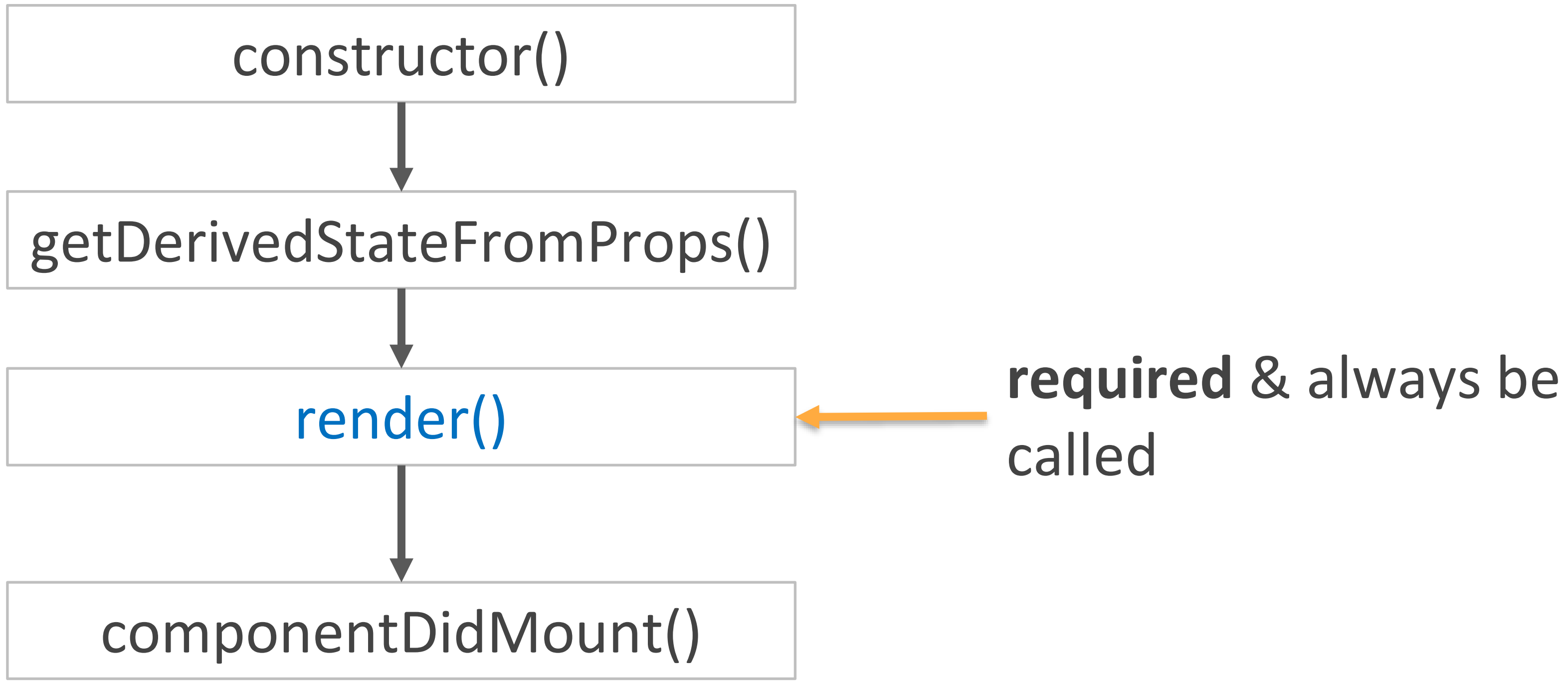
https://www.w3schools.com/react/react_lifecycle.asp

# Component Lifecycle



# Mounting

Putting elements into the DOM



# constructor()

- Called by React
- Set up **initial state**
- **Note:** always start by calling `super(props)`

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      favoriteColor: "red"
    };
  }
  render() {
    return <h1>
       My favorite color is { this.state.favoriteColor }
    </h1>



ReactDOM.render(<Header />, document.querySelector('#root'));
```

Output?

# getDerivedStateFromProps()

- Set the **state** object based on the initial **props**
- Takes **state** as an argument
  - Returns **state** with changes

e.g. favorite color = "**red**"

→ **getDerivedStateFromProps()**

→ favorite color = **favcol** attribute

Output?

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      favoriteColor: "red"
    };
  }
  static getDerivedStateFromProps(props, state) {
    return { favoriteColor: props.favcol }
  }
  // ...
}
ReactDOM.render(<Header favcol="yellow" />,
document.querySelector('#root'));
```

## render()

- Is **required**
- The method that actual **outputs HTML** to the DOM

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      favoriteColor: "red"
    };
  }
  render() {
    return <h1>
       My favorite color is { this.state.favoriteColor }
    </h1>
  }
  // ...
}

ReactDOM.render(<Header />, document.querySelector('#root'));
```

# componentDidMount()

- Is called after the component is **rendered**

→ To run statements that requires the component is already placed to the DOM

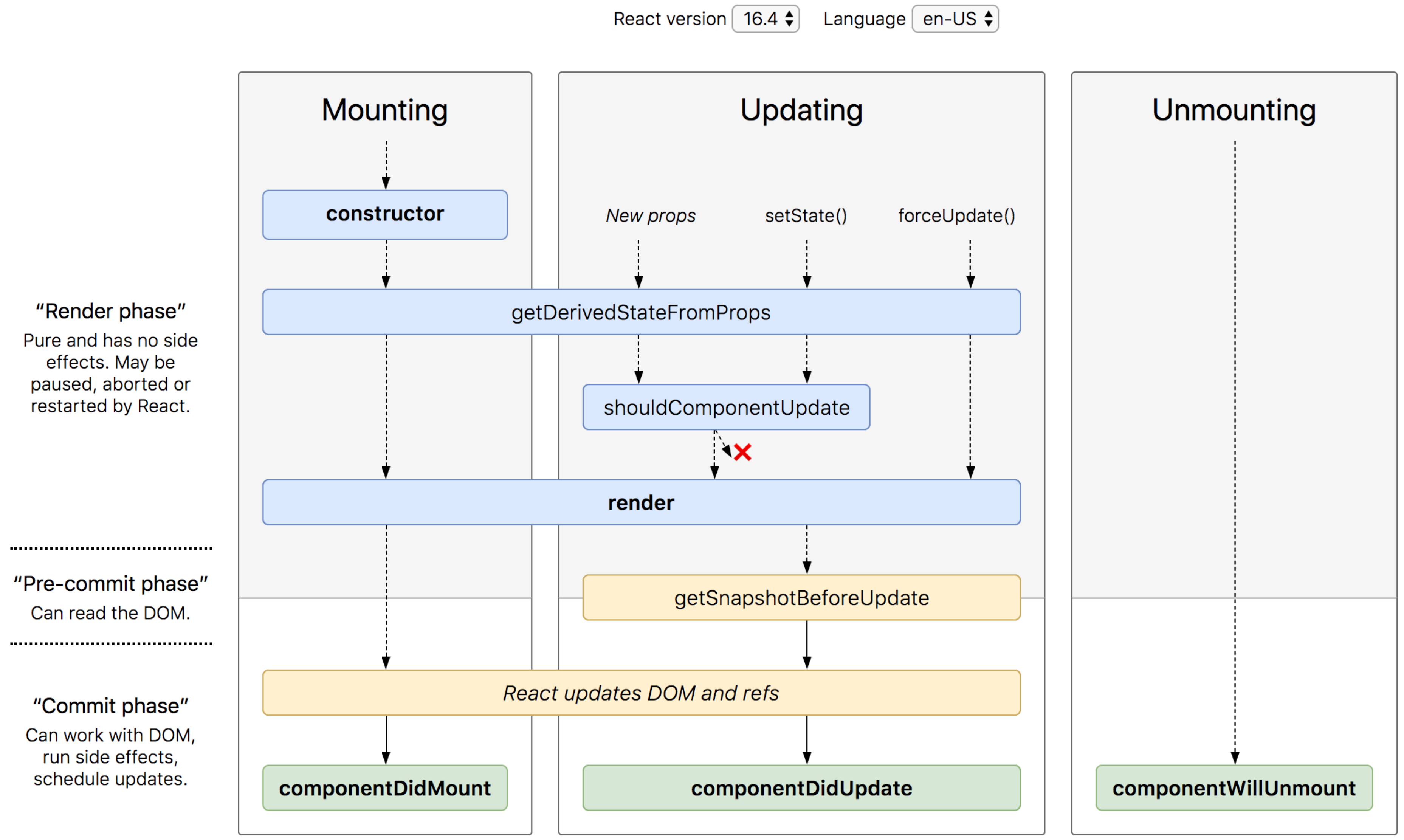
e.g. at first my favorite color is **red**, but give me a second, and it is **yellow** instead

(w3schools)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      favoriteColor: "red"
    };
  }
  componentDidMount() {
     setTimeout(()=>{
      this.setState({ favoriteColor: "yellow" });
     }, 1000);
  }
}
```

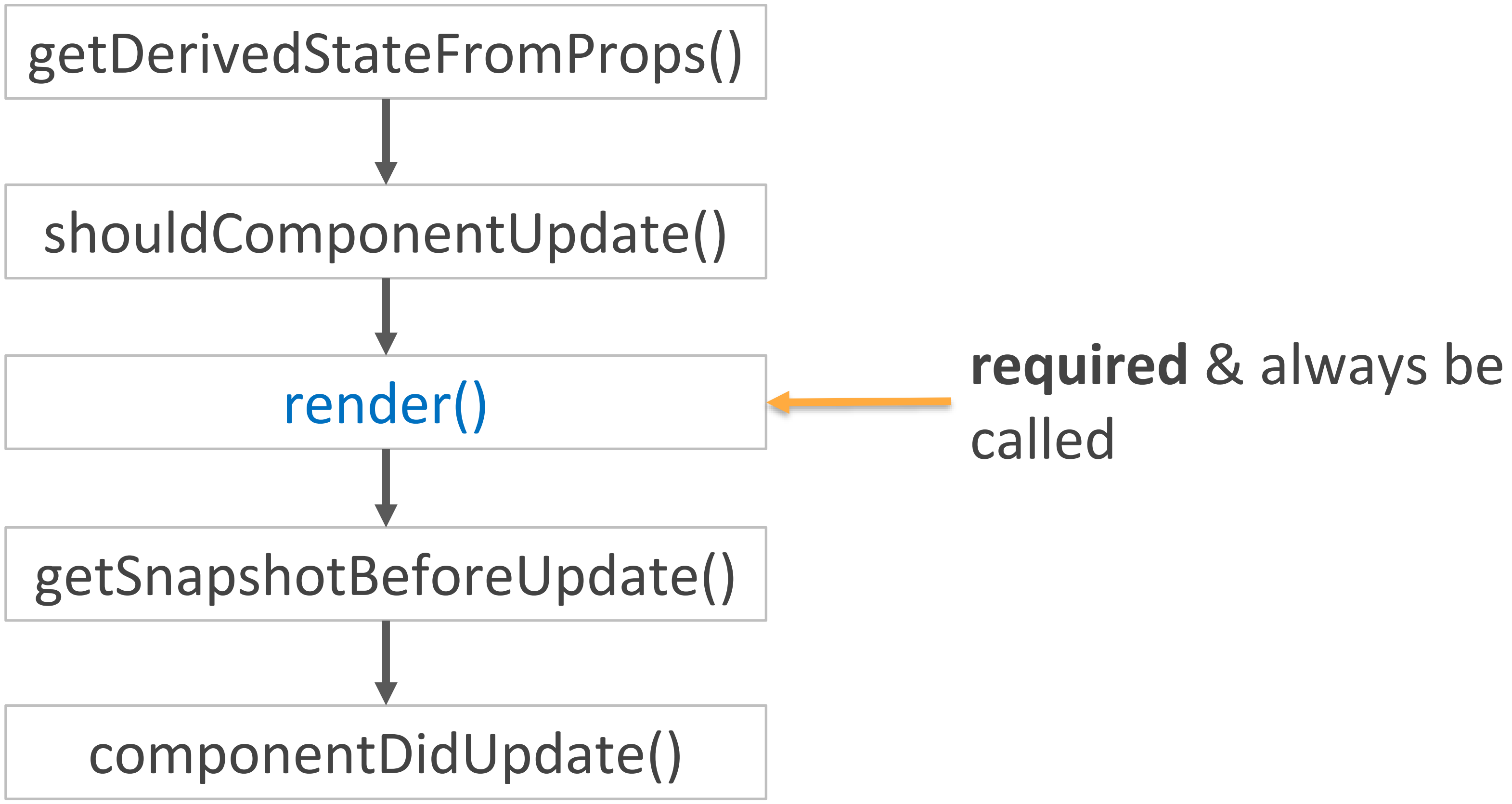
Output?

# Component Lifecycle



# Updating

Any change in state or props → update component


getDerivedStateFromProps()
shouldComponentUpdate()
render()
getSnapshotBeforeUpdate()
componentDidUpdate()
**required** & always be called

# getDerivedStateFromProps()

- Set the **state** object based on the initial **props**

e.g. button click

→ favorite color = **blue**

**BUT**
**`getDerivedStateFromProps()`**

→
favorite color = **`favcol`** attribute

→ rendered as yellow
([w3schools](https://www.w3schools.com))

Output?

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      favoriteColor: "red"
    };
  }
  static getDerivedStateFromProps(props, state) {
    return {favoriteColor: props.favcol}
  }
  changeColor = () => {
    this.setState({favoriteColor: "blue"})
  }
  render() {
    return <div>
    <h1>My favorite color is { this.state.favoriteColor }</h1>
    <button onClick={this.changeColor}>Change color</button>
    </div>;
  }
}
ReactDOM.render(<Header favcol="yellow" />,
document.querySelector('#root'));
```

# shouldComponentUpdate()

- Returns whether React should continue with the rendering or not

e.g. stop the component from rendering at any update (w3schools)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      favoriteColor: "red"
    };
  }
  shouldComponentUpdate() {
    return false;
  }
  changeColor = () => {
    this.setState({favoriteColor: "blue"})
  }
  render() {
    return <div>
    <h1>My favorite color is { this.state.favoriteColor }</h1>
    <button onClick={this.changeColor}>Change color</button>
    </div>;
  }
}
```

Output?

# render()

- to **re-render** the HTML to the DOM with new changes

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      favoriteColor: "red"
    };
  }
  changeColor = () => {
    this.setState({favoriteColor: "blue"})
  }
  render() {
    return <div>
      <h1>My favorite color is { this.state.favoriteColor }</h1>
      <button onClick={this.changeColor}>Change color</button>
        v>;



ReactDOM.render(<Header />, document.querySelector('#root'));
```

Output?

# getSnapshotBeforeUpdate()

In this method, we have access to the props & state before the update
→ We can check previous values after the update

e.g. Feature: Undo – Redo

- Must also include **componentDidUpdate()** method

## `componentDidUpdate()`

Is called after the component is **updated** in the DOM

## getSnapshotBeforeUpdate()

e.g.

- Mounting: favorite color = "**red**".

- Mounted: a timer changes the state (after 1s), favorite color = "**yellow**".

→ This triggers the update phase

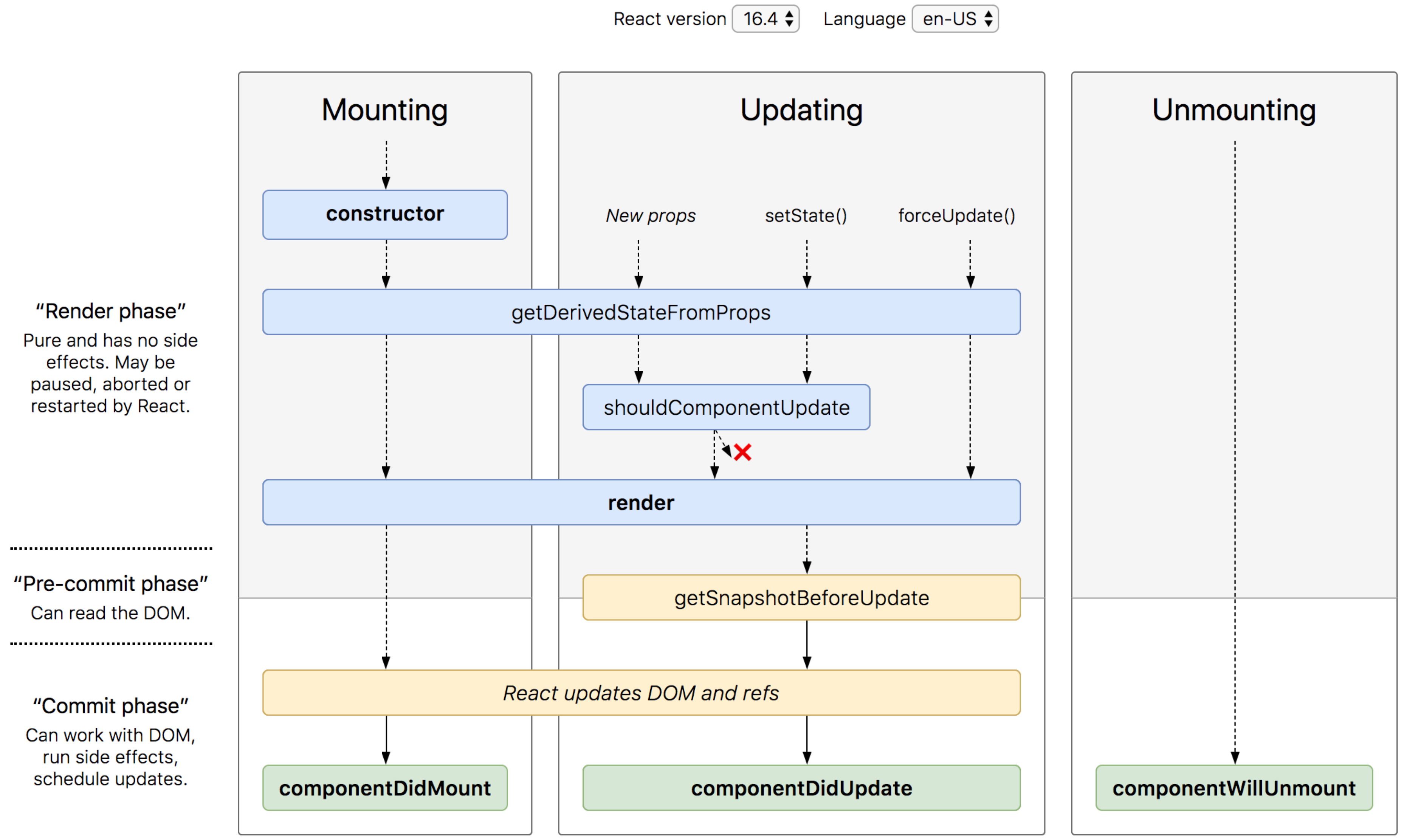
→ **`getSnapshotBeforeUpdate()`** writes a message to div1

→ **`componentDidUpdate()`** writes a message to div2

(w3schools)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoriteColor: "red" };
  }
  componentDidMount() {
    setTimeout(()=>{
      this.setState({ favoriteColor: "yellow" });
    }, 1000);
  }

  getSnapshotBeforeUpdate(prevProps, prevState) {
    document.querySelector("#div1").innerHTML =
    "Before the update, the favorite was "+prevState.favoriteColor;
  }

  componentDidUpdate() {
    document.querySelector("#div2").innerHTML =
    "The updated favorite is " + this.state.favoriteColor;
  }

  render() {
    return <div>
      <h1>My favorite color is { this.state.favoriteColor }</h1>
      <div id="div1"></div>
      <div id="div2"></div>
    </div>;
  }
}
ReactDOM.render(<Header />, document.querySelector('#root'));
```

# Component Lifecycle



## Unmounting

When a component is removed from the DOM

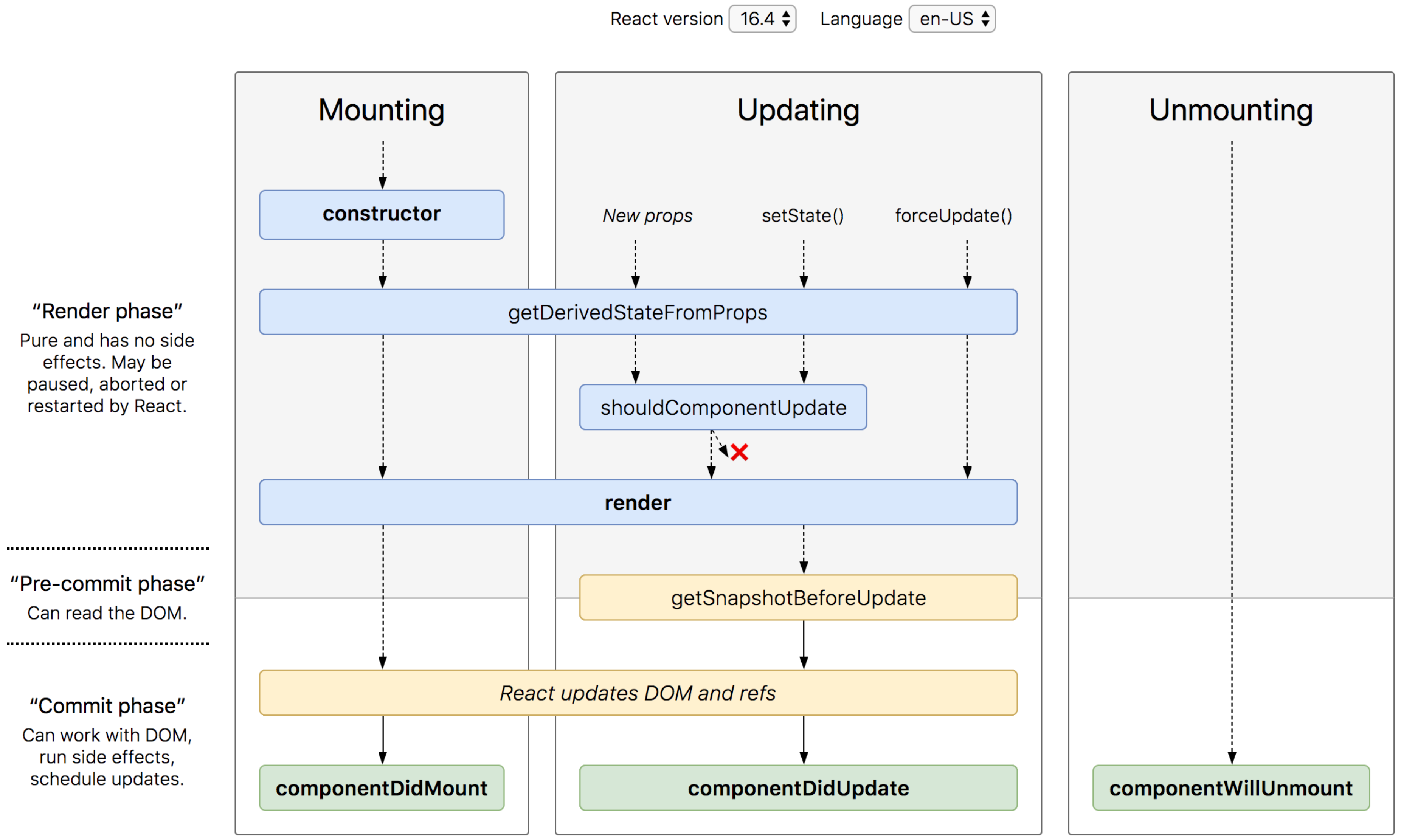
componentWillUnmount()

# componentWillUnmount()

e.g. click button to delete child

```
class Container extends React.Component {
    constructor() {
        super();
        this.state = { show: true };
    }
    hide = () => {
        this.setState({ show: false });
    }
    render() {
        let child;
        if (this.state.show) {
            child = <Child />;
        }
        return <div>
            { child }
            <button onClick={this.hide}>Hide</button>
        </div>;
    }
}

class Child extends React.Component {
    componentWillUnmount() {
        alert("The component named Child is about to be unmounted.")
    }
    render() {
        return <h1>Hello World!</h1>;
    }
}
ReactDOM.render(<Container />, document.querySelector("#root"));
```

# Component Lifecycle

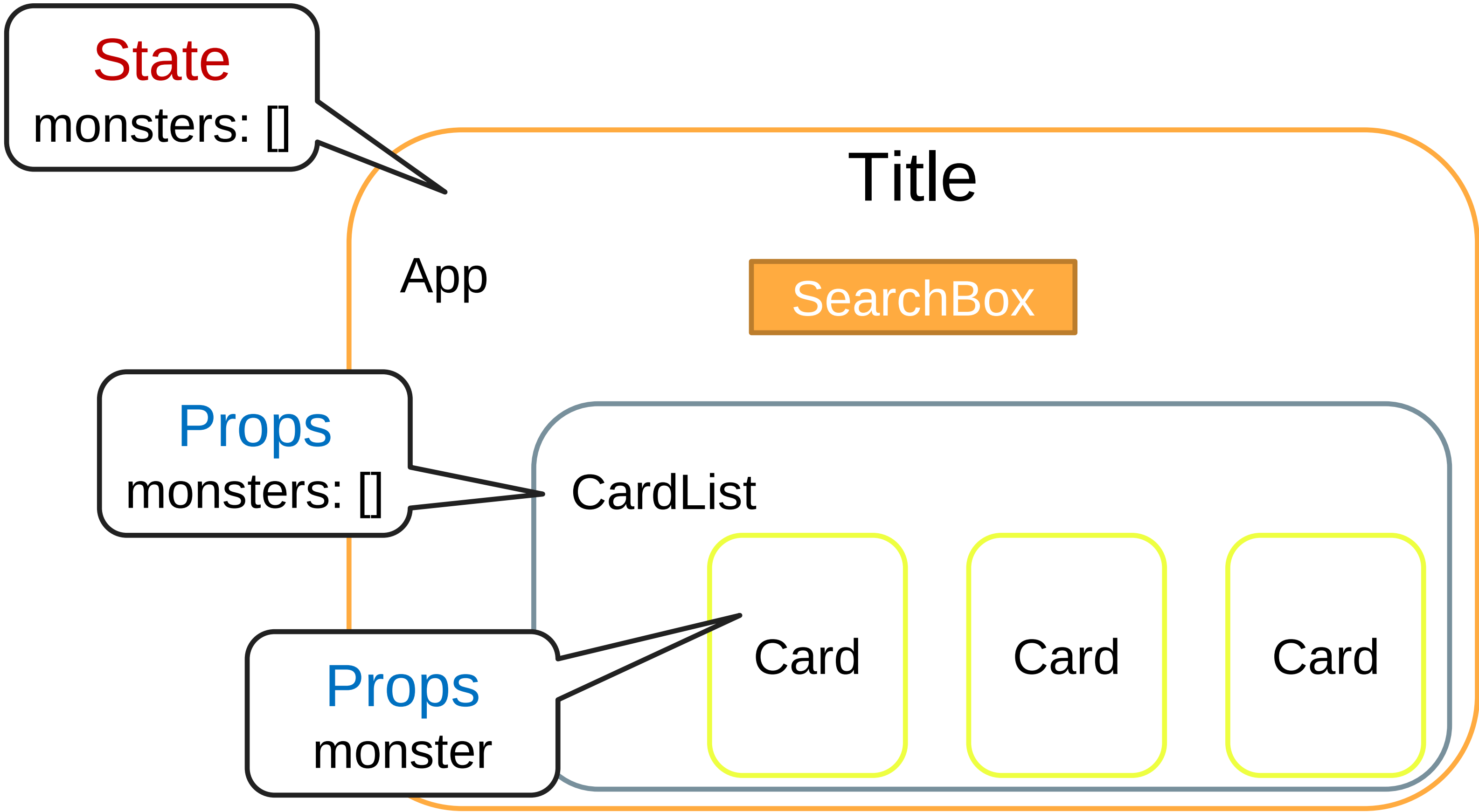


## React events

## Example: monsters

- **Search field:**
  - **State**: keyword
  - **Events**: input/ change keyword

# Example: monsters



## Native event

- HTML events: something DOM give us to interact with user events

```
<input type="text" onchange="" />
```

More about onchange:

https://www.w3schools.com/jsref/event_onchange.asp

# React events

HTML events: **`onclick`, `onchange`, `onmouseover`**, etc.

**Adding events:**

- React events: written in camelCase
  - `onclick` → `onClick`
- React event handlers: written inside curly braces
  - `onclick="shoot()"` → `onClick={shoot}`

React

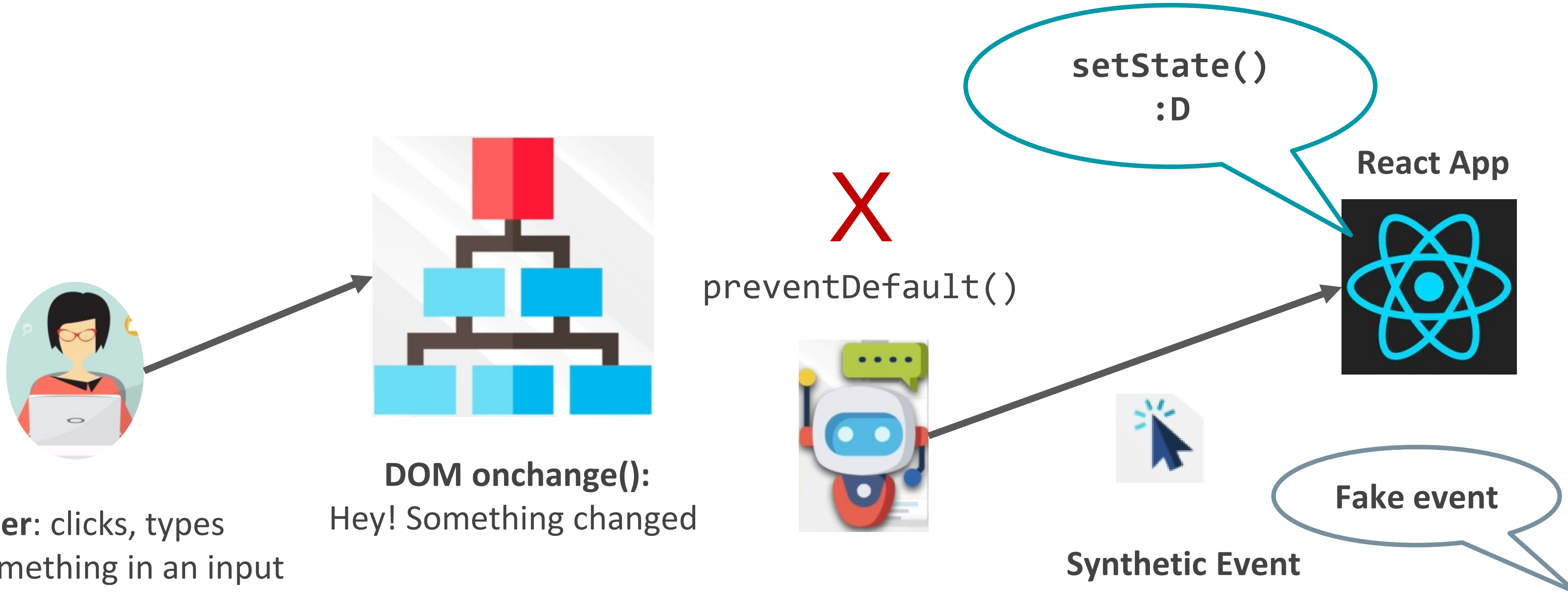
```
<button onClick={shoot}>Take the Shot!</button>
```

HTML

```
<button onclick="shoot()">Take the Shot!</button>
```

# SyntheticEvent

HTML events: **`onclick`**, **`onchange`**, **`onmouseover`**, etc.



**User**: clicks, types something in an input

**DOM onchange():**
Hey! Something changed

**Synthetic Event**

Hey! there was an event on the DOM, what should we do?

## Passing arguments

**Make an anonymous arrow function**

e.g. Send “Goal” as a parameter to the shoot function

```
class Football extends React.Component {
    shoot = (a) => {
        alert(a);
    }

    render() {
        return <button onClick={() => this.shoot("Goal")}>Take the shot!</button>
    }
}

ReactDOM.render(<Football />, document.querySelector("#root"));
```

# React Event Object

Event handlers have access to the React event that triggered the function

- **Without** arrow function, the React event object is sent *automatically*
- **With** arrow function, you have to send the event argument *manually*

```
class Football extends React.Component {
    shoot = (a) => {
        alert(a);
    }
    render() {
        return <button onClick={(event) => this.shoot("Goal", event)}>Take the
shot!</button>
    }
}
ReactDOM.render(<Football />, document.querySelector("#root"));
```

# Example: monsters

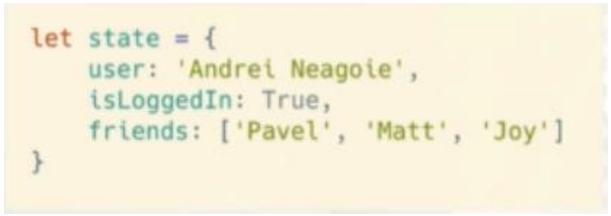
***[code example]***

- **Search field:**
  - **State**: keyword
  - **Events**: input/ change keyword

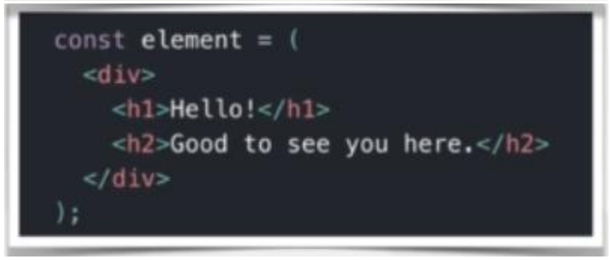
# Recall: Unidirectional data flow

Virtual DOM

State

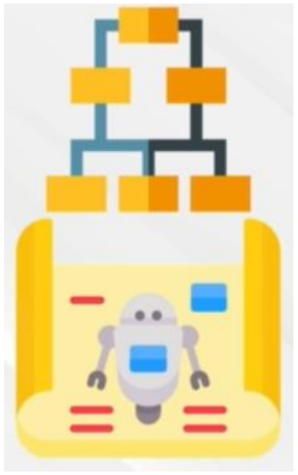


```
let state = {
    user: 'Andrei Neagoie',
    isLoggedIn: True,
    friends: ['Pavel', 'Matt', 'Joy']
}
```

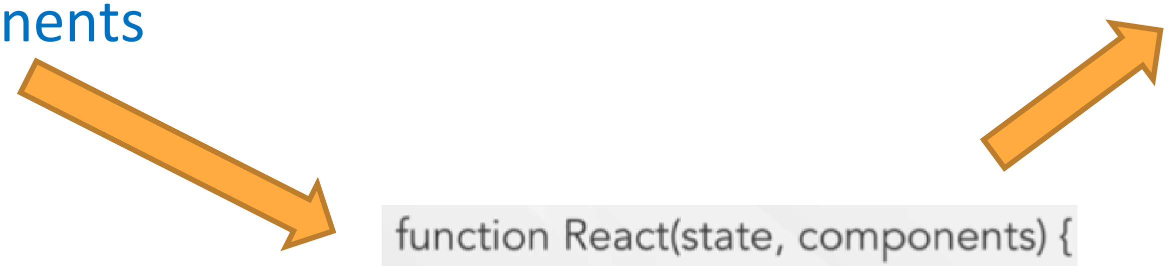


```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

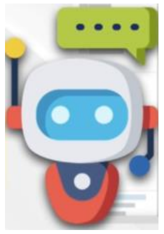
JSX



Components

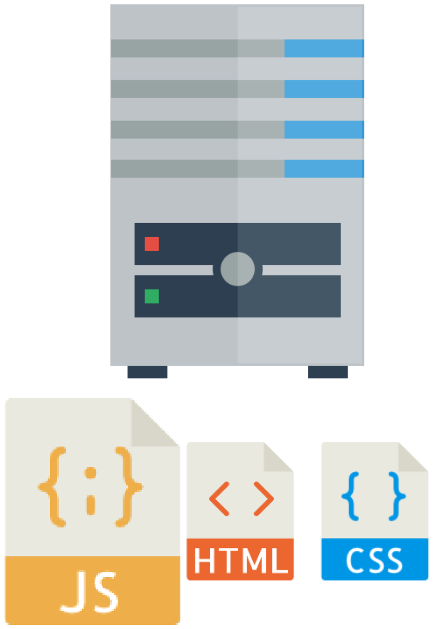


```
function React(state, components) {

}
```



## React routes

# The birth of SPA

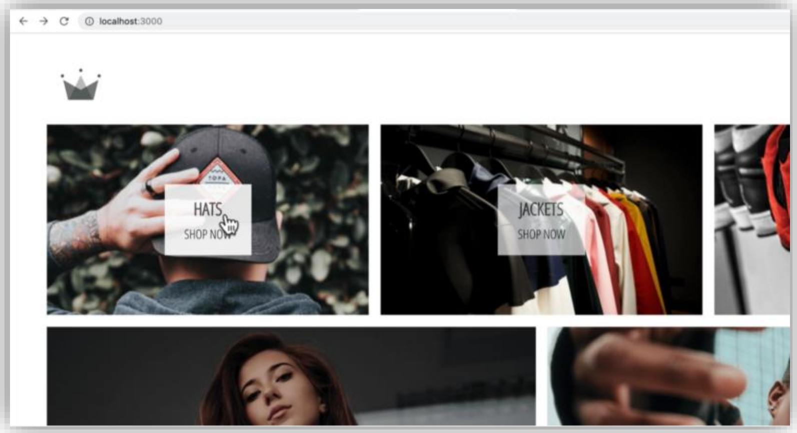


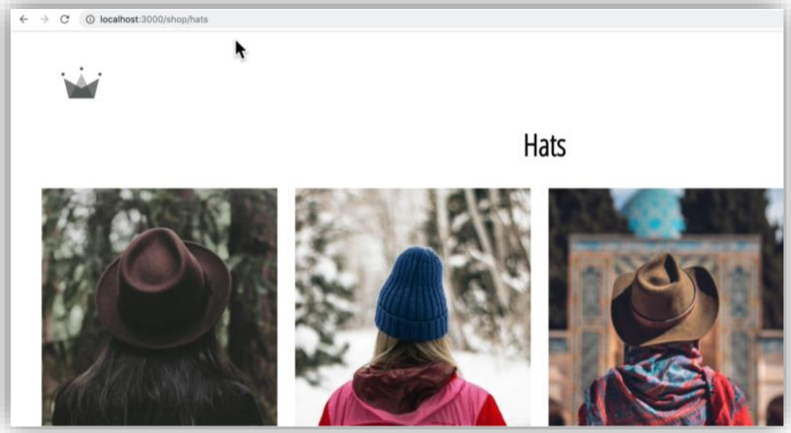
## Problem to solve

Single Page App (SPA) → **NO default browser navigation**
(click on links → get new HTML file → update URL)

→ **Back/ Forward**: not work
→ **Refresh**: to home page, may lose data

**localhost:3000** → **localhost:3000/shop/hats**





# React router

→ Solution: **browser history API** : mimic the URL

Remember, React is **just a UI library** – no pre-built routing

→ Implement our own

→ **Use a library**

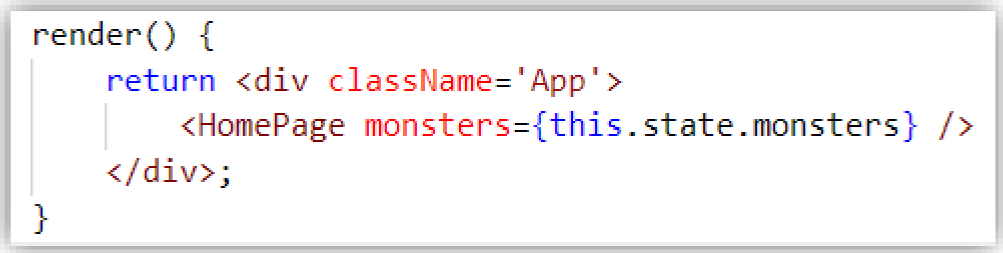
**react-router-dom**

Documentation: https://reactrouter.com/

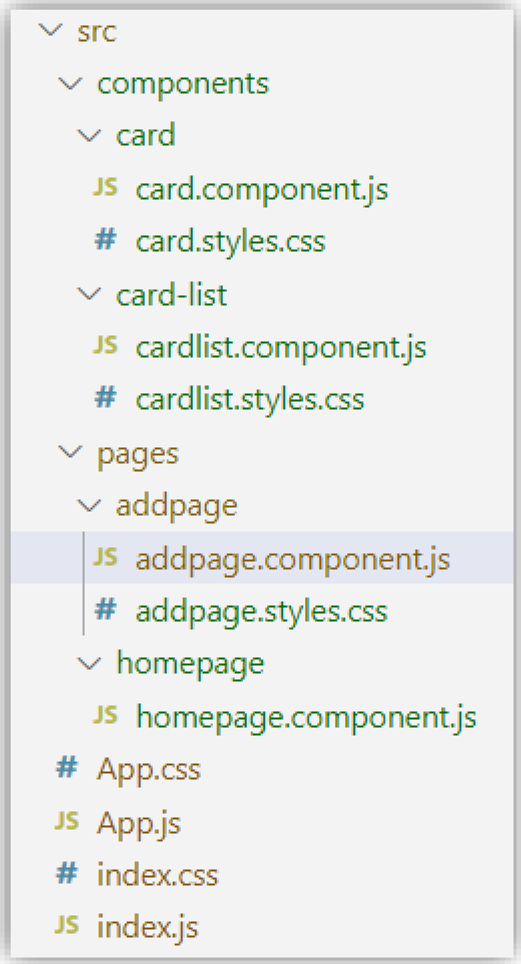
# Example: monsters: my 2nd page

*[code example]*

- Refactor App → **HomePage**
- **AddPage**: add new monster



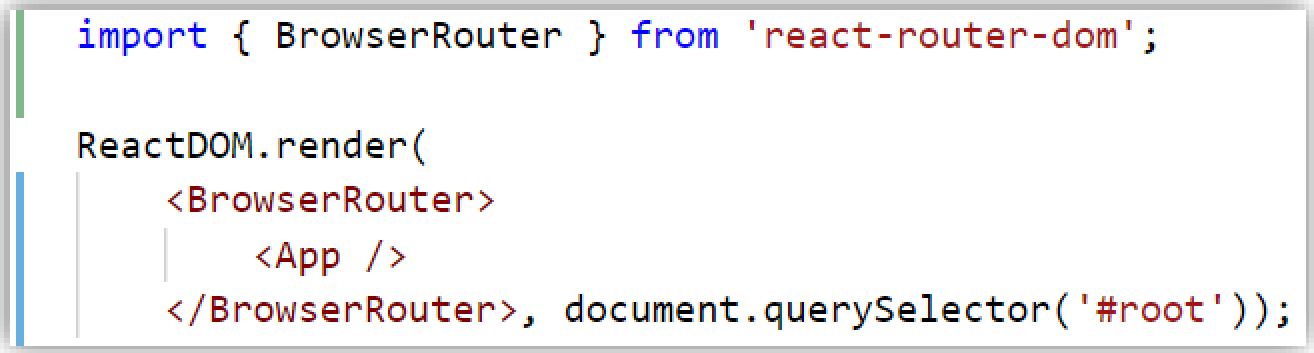
```
render() {
    return <div className='App'>
        <HomePage monsters={this.state.monsters} />
    </div>;
}
```



```
src
  components
    card
      card.component.js
      card.styles.css
    card-list
      cardlist.component.js
      cardlist.styles.css
  pages
    addpage
      addpage.component.js
      addpage.styles.css
    homepage
      homepage.component.js
  App.css
  App.js
  index.css
  index.js
```

## <BrowserRouter>

***[code example]***

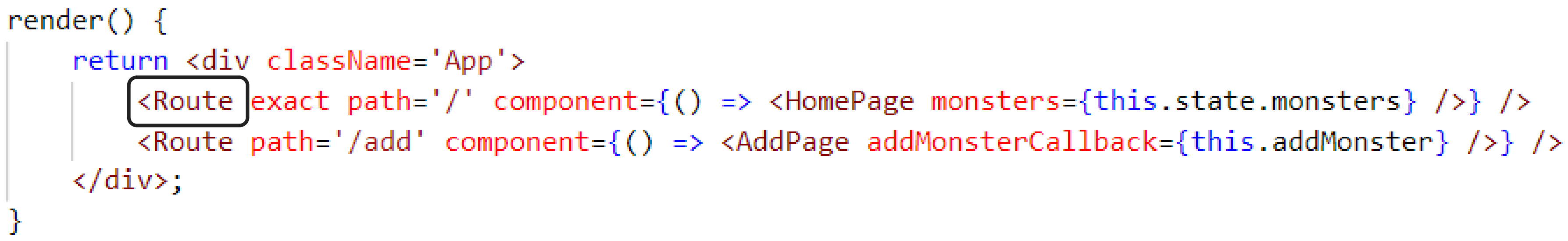


```
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
    <BrowserRouter>
        <App />
    </BrowserRouter>, document.querySelector('#root'));
```

- `index.js`:
  - wrap around **App** with **<BrowserRouter>** component
  - All the functionality of routing provided by this library now in order for App

# <Route>

***[code example]***

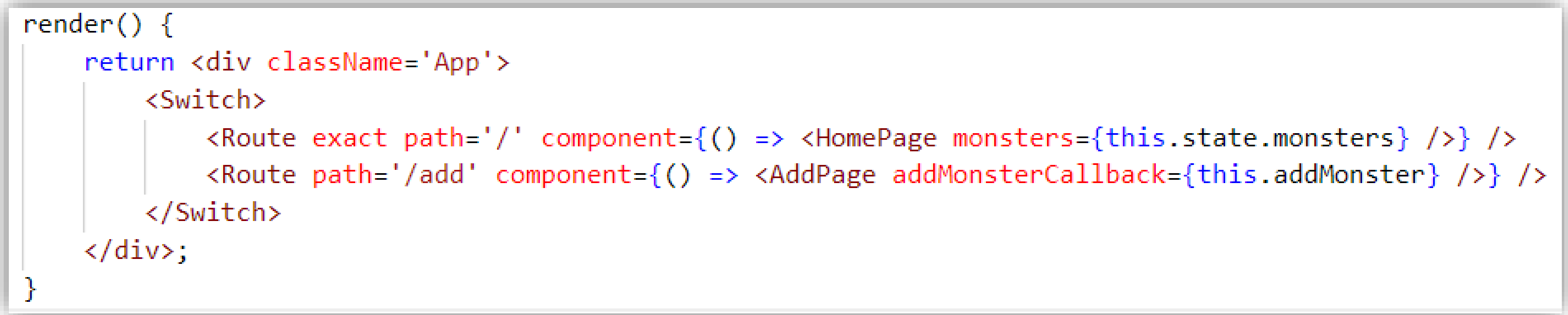


```
render() {
    return <div className='App'>
        <Route exact path='/' component={() => <HomePage monsters={this.state.monsters} />} />
        <Route path='/add' component={() => <AddPage addMonsterCallback={this.addMonster} />} />
    </div>;
}
```

- **<Route** [exact] path component**>**
  - *component*: component we want to render
  - *path*: string: url
  - *exact*: boolean: true = path must be exactly matched

# <Switch>

*[code example]*

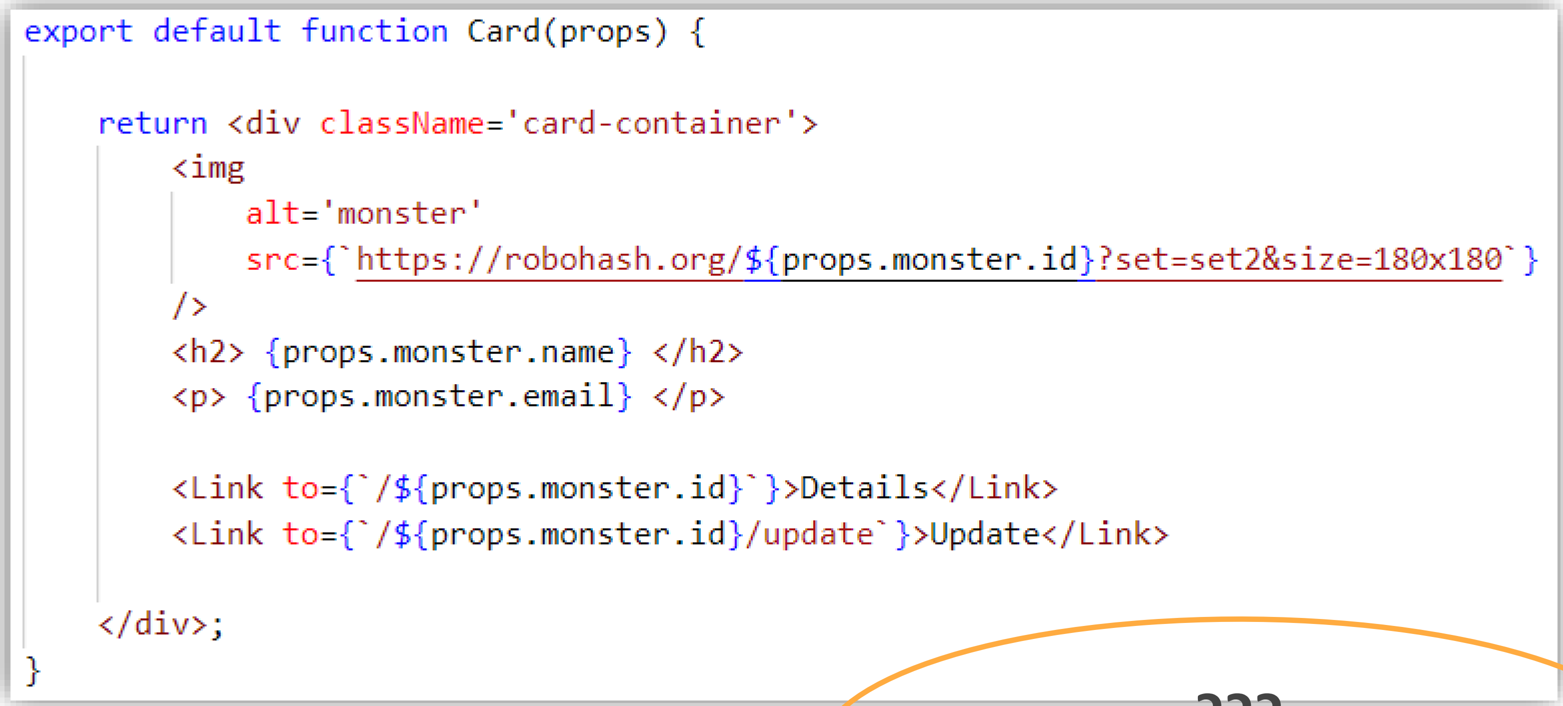


```
render() {
    return <div className='App'>
        <Switch>
            <Route exact path='/' component={() => <HomePage monsters={this.state.monsters} />} />
            <Route path='/add' component={() => <AddPage addMonsterCallback={this.addMonster} />} />
        </Switch>
    </div>;
}
```

- **<Switch>** Only first matched route & NOTHING more

# <Link>

*[code example]*



```
export default function Card(props) {

    return <div className='card-container'>
        <img
            alt='monster'
            src={`https://robohash.org/${props.monster.id}?set=set2&size=180x180`}
        />
        <h2> {props.monster.name} </h2>
        <p> {props.monster.email} </p>

        <Link to={`/${props.monster.id}`}>Details</Link>
        <Link to={`/${props.monster.id}/update`}>Update</Link>

    </div>;
}
```

- Create & Display the **<a>** link

**???**
**Why not just <a> tag**

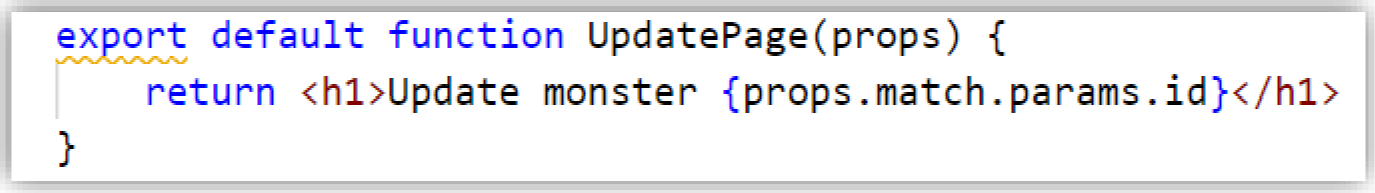
## <NavLink>

***[code example]***

- <Link> with active CSS class based on the route
- Use for menu

# <Example: monsters: params

***[code example]***



```
export default function UpdatePage(props) {
    return <h1>Update monster {props.match.params.id}</h1>
}
```

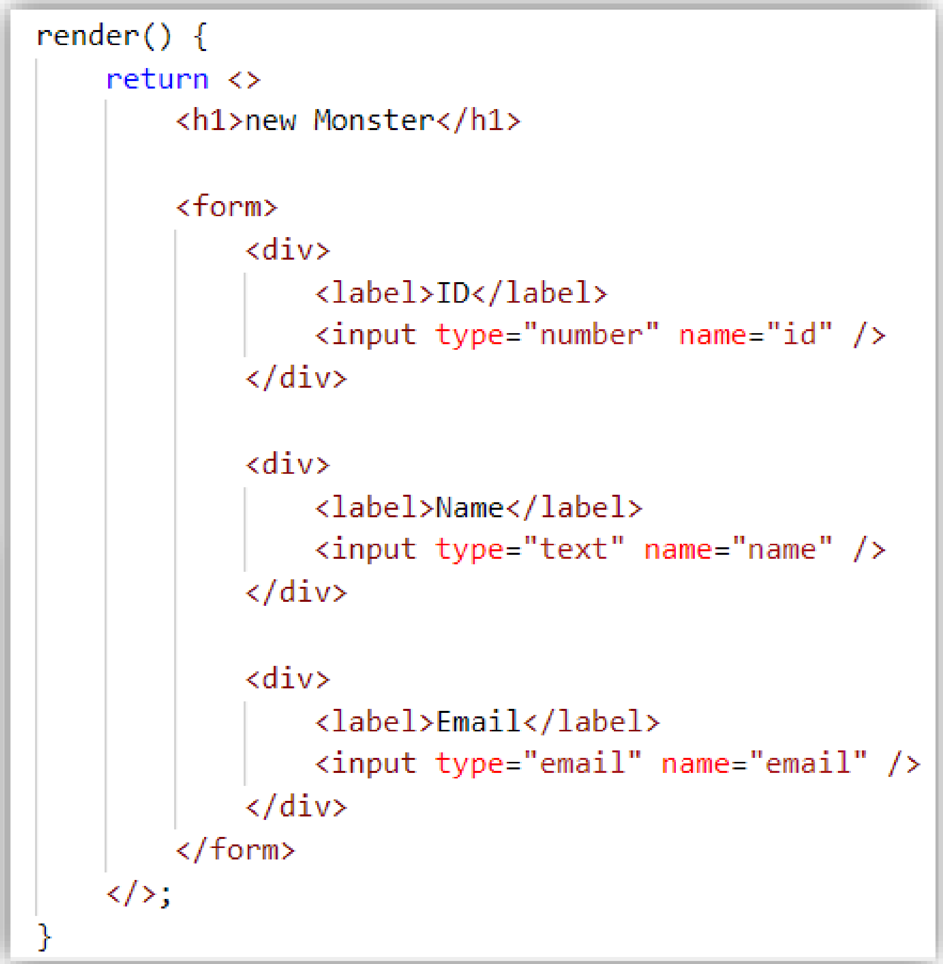
`props.match.params.paramName`

## React forms (optional)

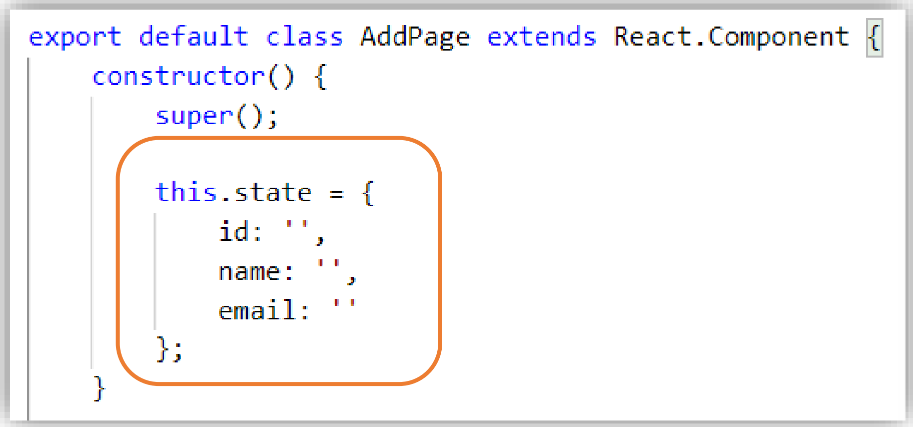
# Example: monsters: new monster

***[code example]***

- state = form data



```
render() {
    return <>
        <h1>new Monster</h1>

        <form>
            <div>
                <label>ID</label>
                <input type="number" name="id" />
            </div>

            <div>
                <label>Name</label>
                <input type="text" name="name" />
            </div>

            <div>
                <label>Email</label>
                <input type="email" name="email" />
            </div>
        </form>
    </>;
}
```

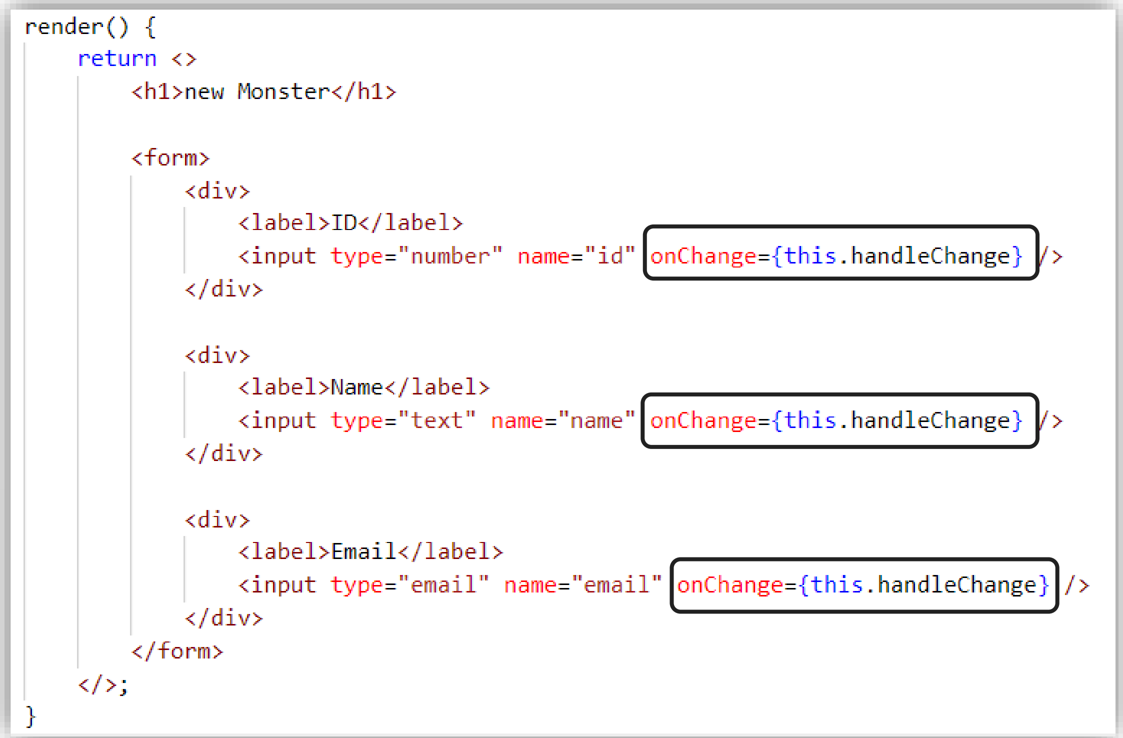


```
export default class AddPage extends React.Component {
    constructor() {
        super();

        this.state = {
            id: '',
            name: '',
            email: ''
        };
    }
```

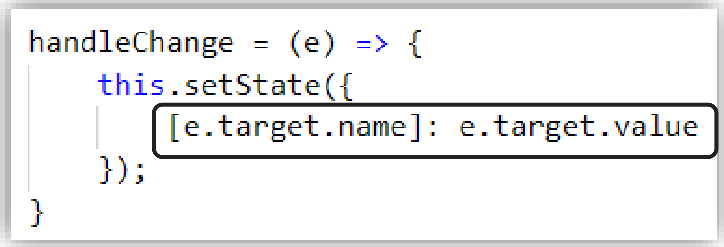
# Example: monsters: handleChange()

***[code example]***

- Update corresponding state field with new value



```
render() {
    return <>
        <h1>new Monster</h1>

        <form>
            <div>
                <label>ID</label>
                <input type="number" name="id" onChange={this.handleChange} />
            </div>

            <div>
                <label>Name</label>
                <input type="text" name="name" onChange={this.handleChange} />
            </div>

            <div>
                <label>Email</label>
                <input type="email" name="email" onChange={this.handleChange} />
            </div>
        </form>
    </>;
}
```

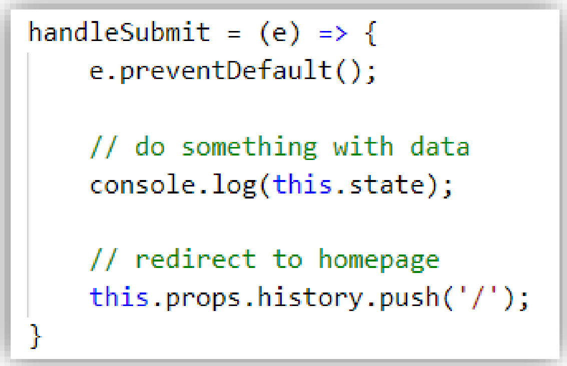


```
handleChange = (e) => {
    this.setState({
        [e.target.name]: e.target.value
    });
}
```

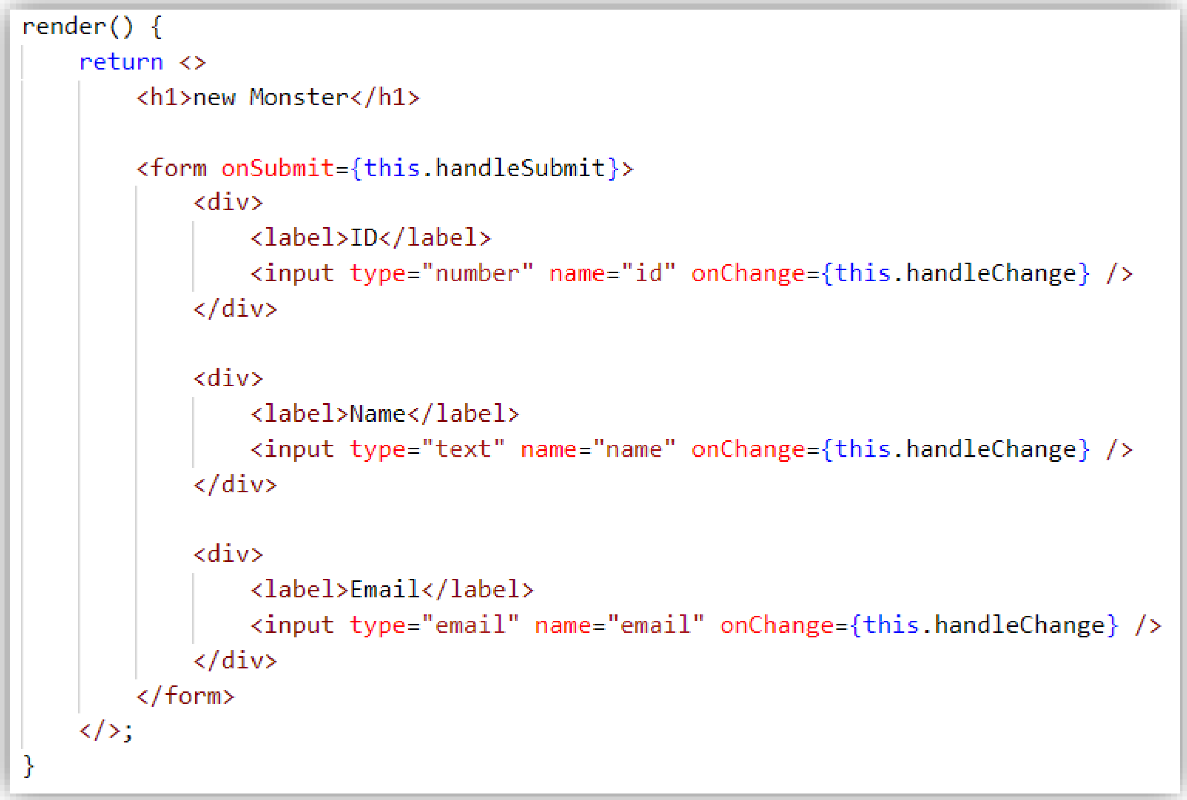
# Example: monsters: onSubmit()

*[code example]*

- **Note**: Prevent default submit behavior (refresh)



```
handleSubmit = (e) => {
    e.preventDefault();

    // do something with data
    console.log(this.state);

    // redirect to homepage
    this.props.history.push('/');
}
```



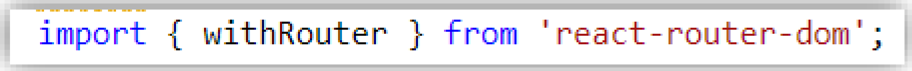
```
render() {
    return <>
        <h1>new Monster</h1>

        <form onSubmit={this.handleSubmit}>
            <div>
                <label>ID</label>
                <input type="number" name="id" onChange={this.handleChange} />
            </div>

            <div>
                <label>Name</label>
                <input type="text" name="name" onChange={this.handleChange} />
            </div>

            <div>
                <label>Email</label>
                <input type="email" name="email" onChange={this.handleChange} />
            </div>
        </form>
    </>;
}
```

**???**
**history**

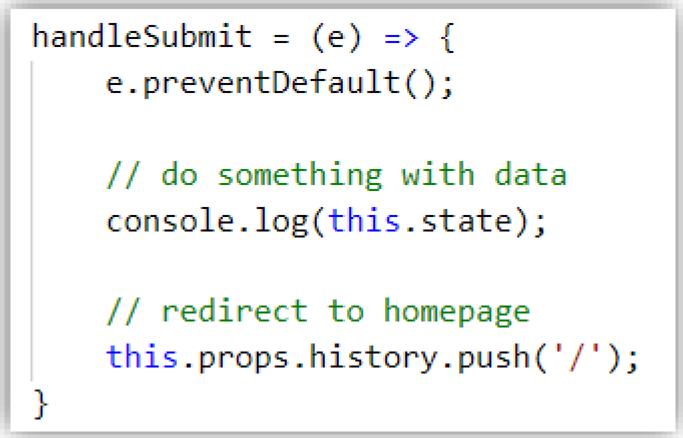
# Example: monsters: withRouter()

***[code example]***

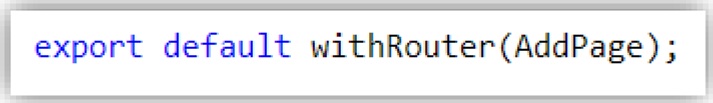
- To access history API → redirect



```
import { withRouter } from 'react-router-dom';
```



```
handleSubmit = (e) => {
    e.preventDefault();

    // do something with data
    console.log(this.state);

    // redirect to homepage
    this.props.history.push('/');
}
```



```
export default withRouter(AddPage);
```

Next week:

**Wrap up!**