Schedule

Note on Assignment 3

Recall:

- React key concepts
- React components

Today:

- CSS in React
- Dynamic content
- Using Props
- Fetching data from API
- Component life cycle

Recall: React key concepts

1. Don’t touch the DOM. I’ll do it

React find the best way to change the DOM **automatically**

State: one big JS object describes how our app should look

e.g.

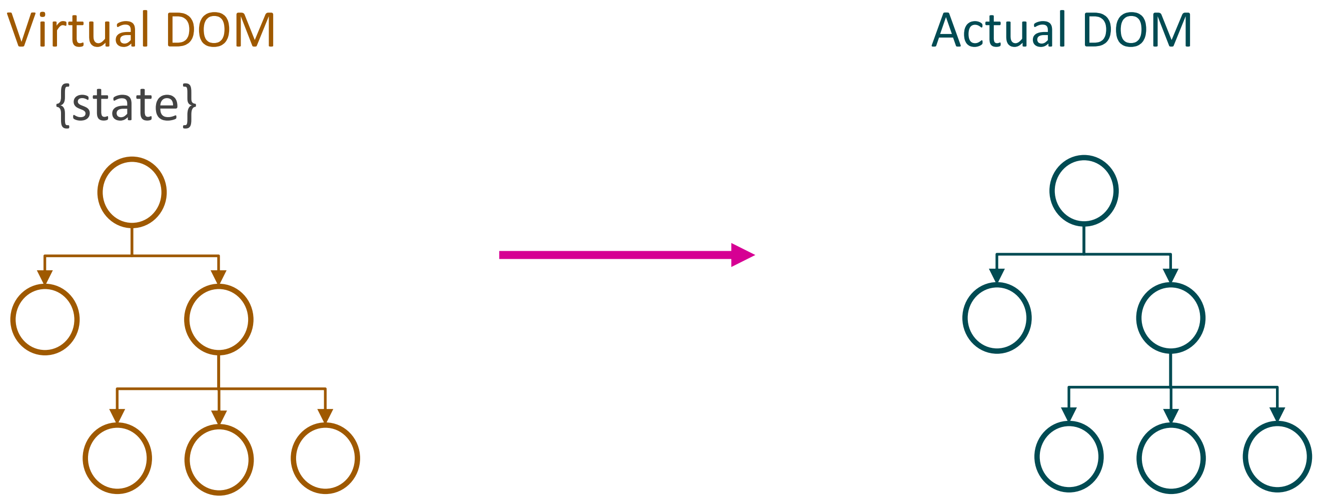
```
{
    loggedIn: false
}
```

$\longrightarrow$

```
{
    loggedIn: true,
    user: {
        name: "Dennis Nguyen",
        friends: [
            "Nguyen Huu Cam", "Nguyen
Thang", "Hieu Nguyen"
        ]
    }
}
```

React virtual DOM

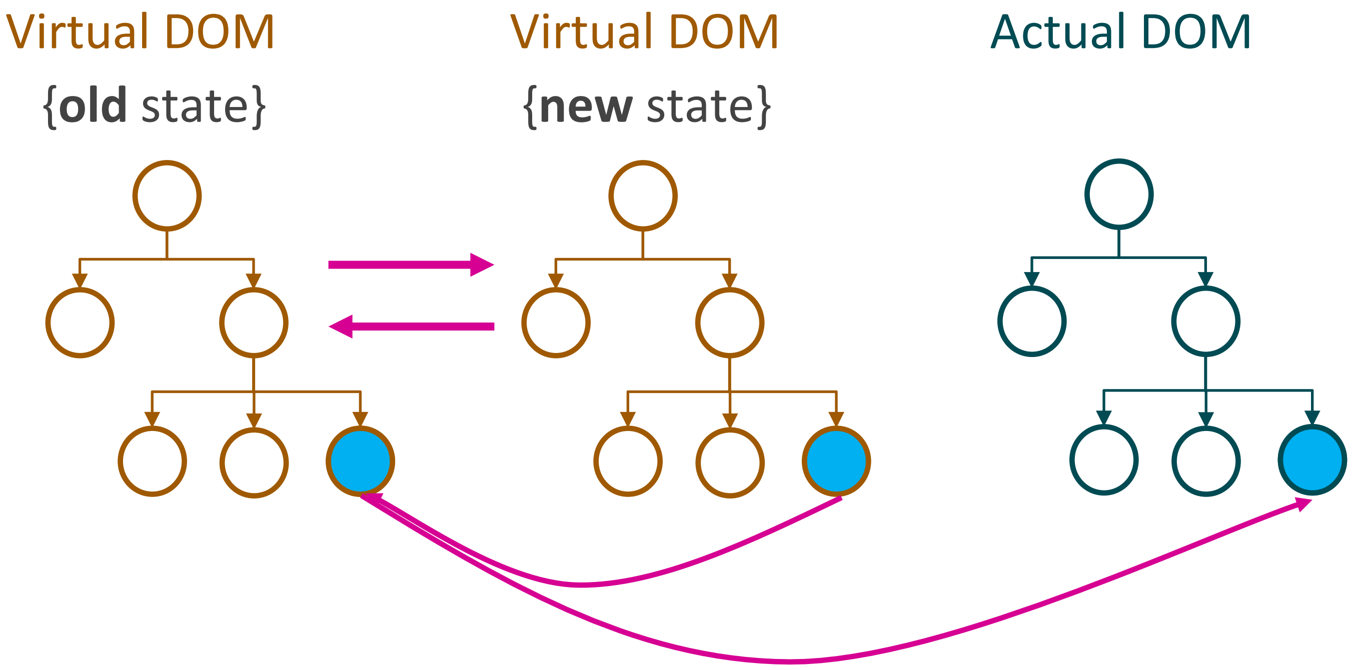
*Hey React, this is the **state** (data) of our app → display it*



React virtual DOM

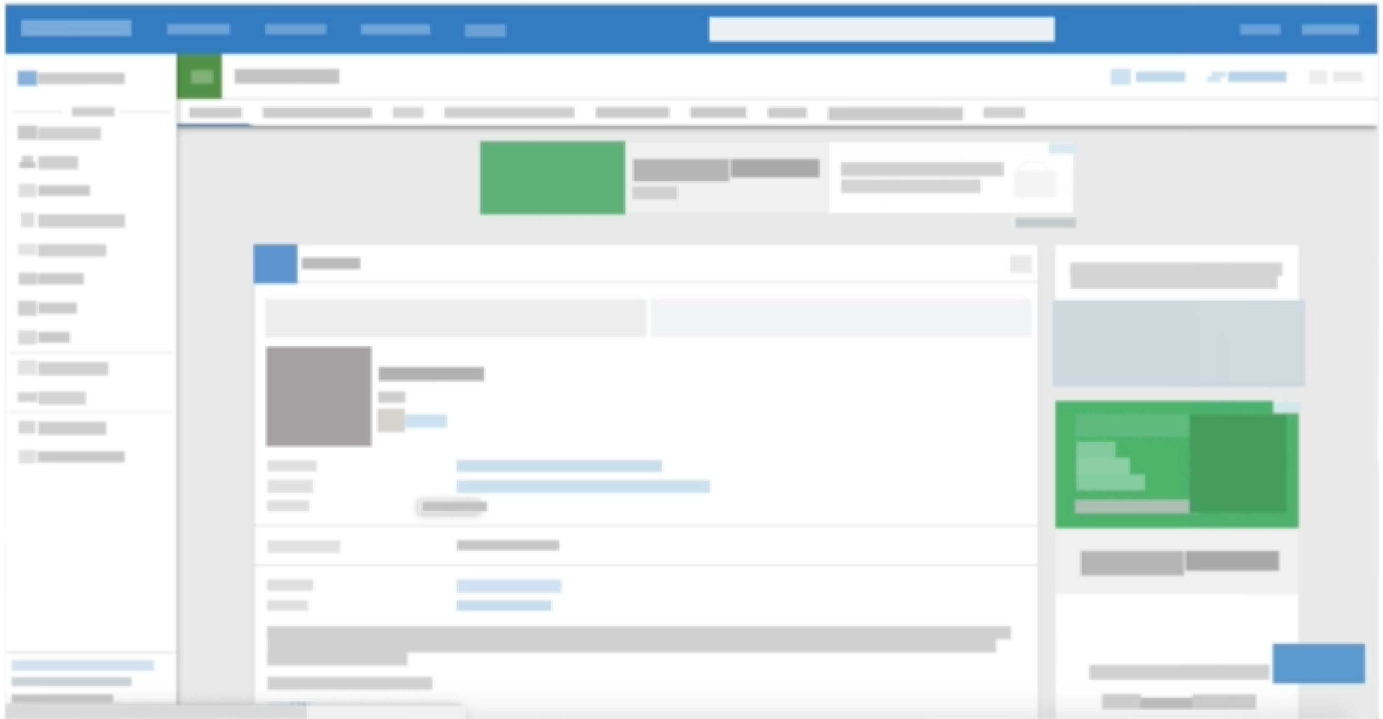
*Hey React, this is the **new state** (data) of our app*

→ make necessary changes to display it

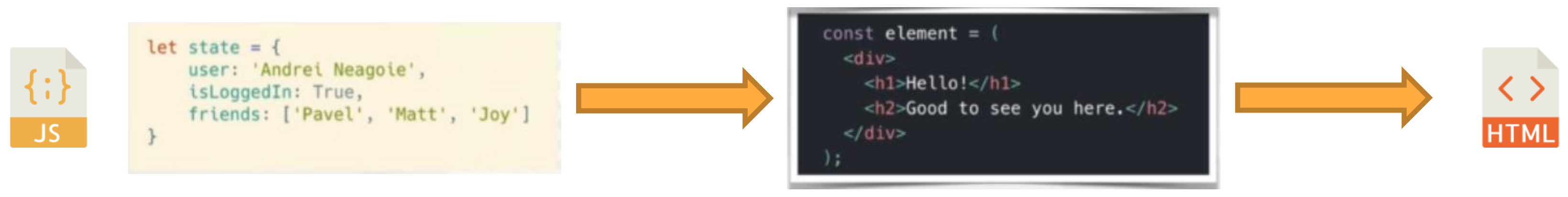


2. Build website like LEGO blocks

- Reusable components
  - e.g. https://material-ui.com/components/buttons/
- Small components – put together → bigger component
- Even move over to different projects



Recall: React Components



```
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>
);
```

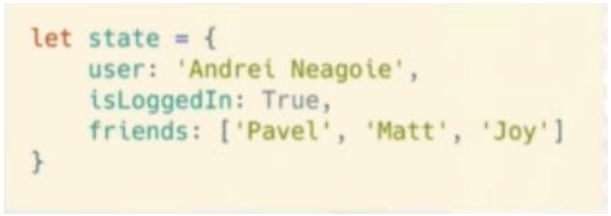
Data

Component

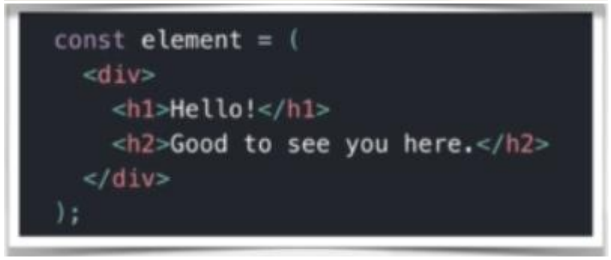
3. Unidirectional data flow

Virtual DOM

State

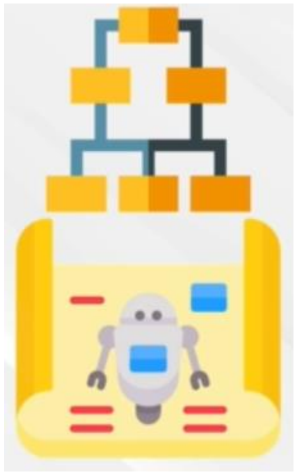


```
let state = {
    user: 'Andrei Neagoie',
    isLoggedIn: True,
    friends: ['Pavel', 'Matt', 'Joy']
}
```

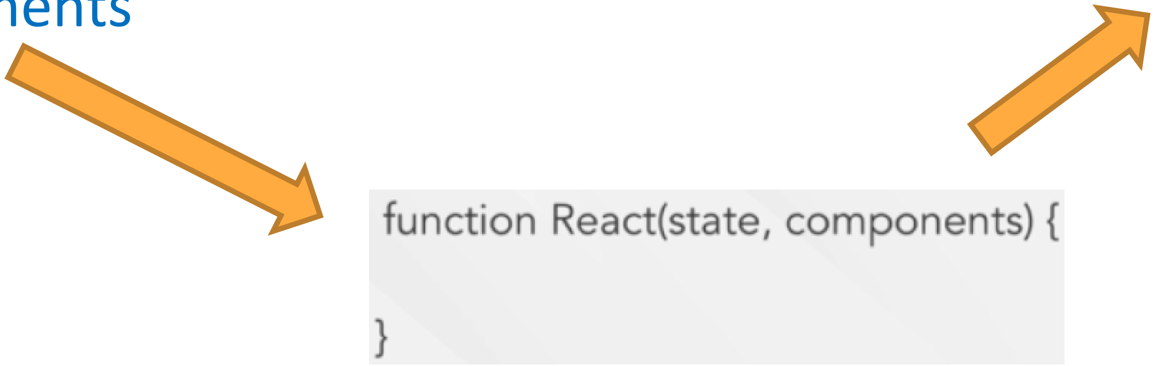


```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

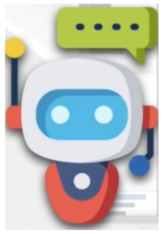
JSX



Components



```
function React(state, components) {

}
```

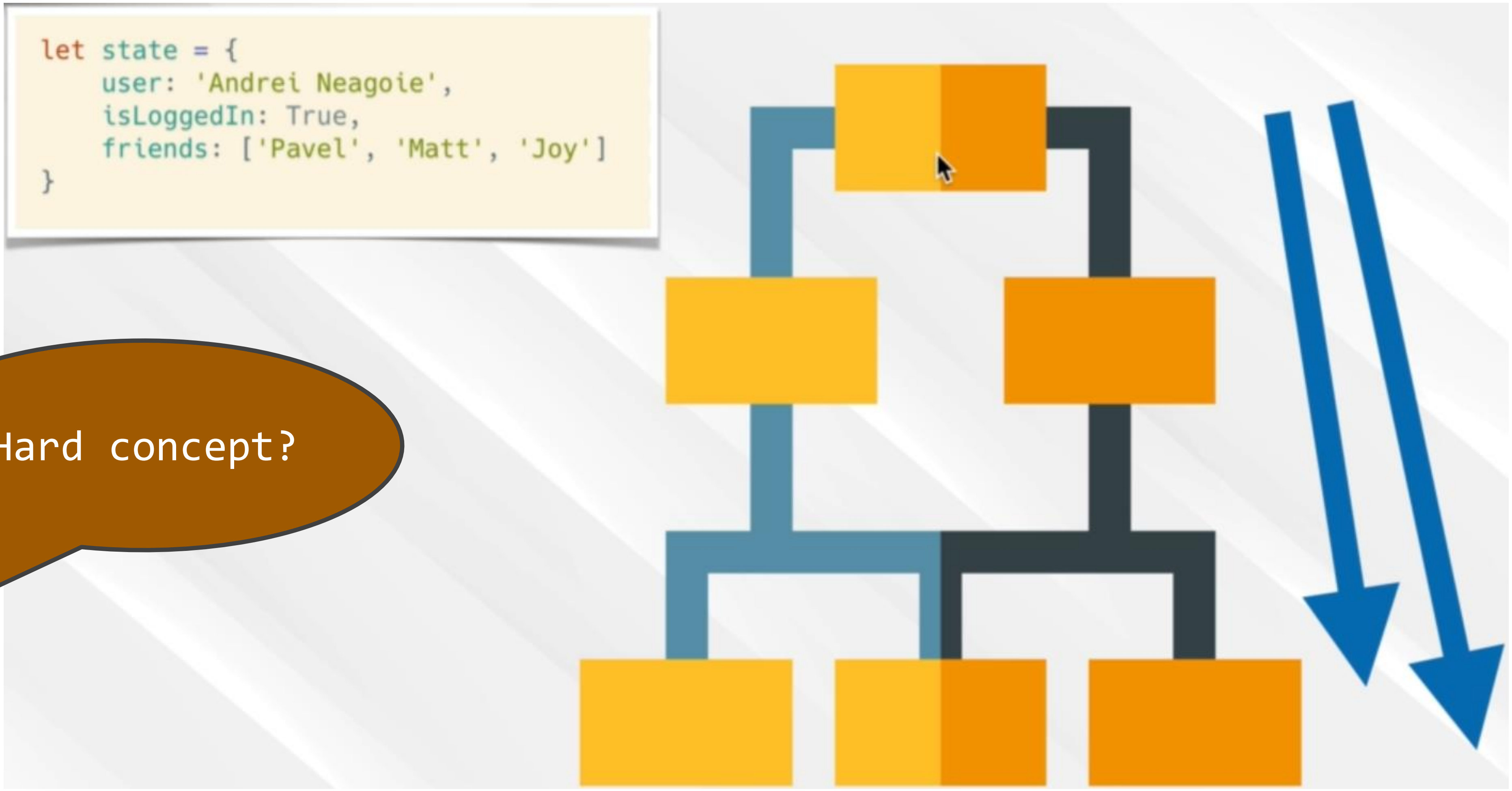


3. Unidirectional data flow

Anytime we want to **change the webpage**

→ **change the state**

3. Unidirectional data flow



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

Hard concept?

3. Unidirectional data flow

Data **never move up**
All the changes can **only trigger down**

4. UI, The rest is up to you

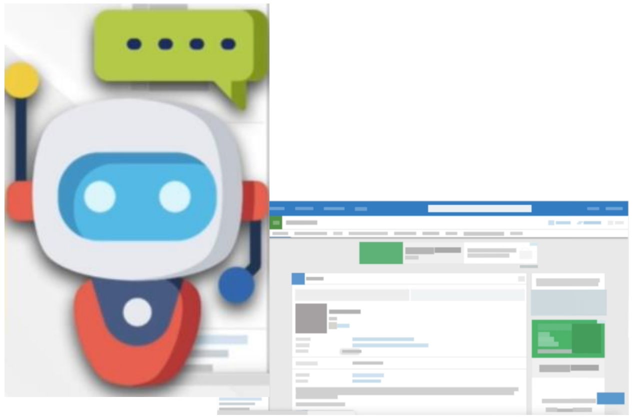
- AngularJS – a framework
- React: UI library (view only)

→ **React everywhere:**

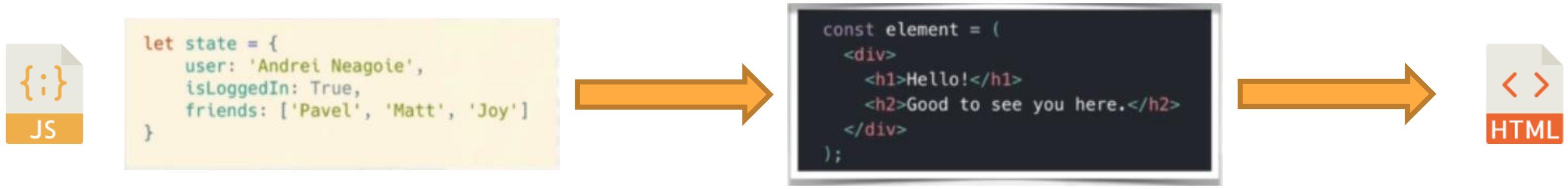
- same principles with JS → build Cross platform

→ (React native – mobile, VR, React desktop, terminal)

React core lib: general robot


React DOM library: specific robot for DOM

Recall: React Components



```
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>
);
```

Data

Component

Function vs Class component

- Function component
  - Function returns HTML
- Class component
  - A lot more functionality (lifecycle)
  - **State**

```
class Car extends React.Component {
    render() {
        return <h2>I am a Car!</h2>;
    }
}
```

```
function Car() {
    return <h2>Hi, I am also a Car!</h2>;
}
```

```
ReactDOM.render(<Car />, document.getElementById('root'));
```

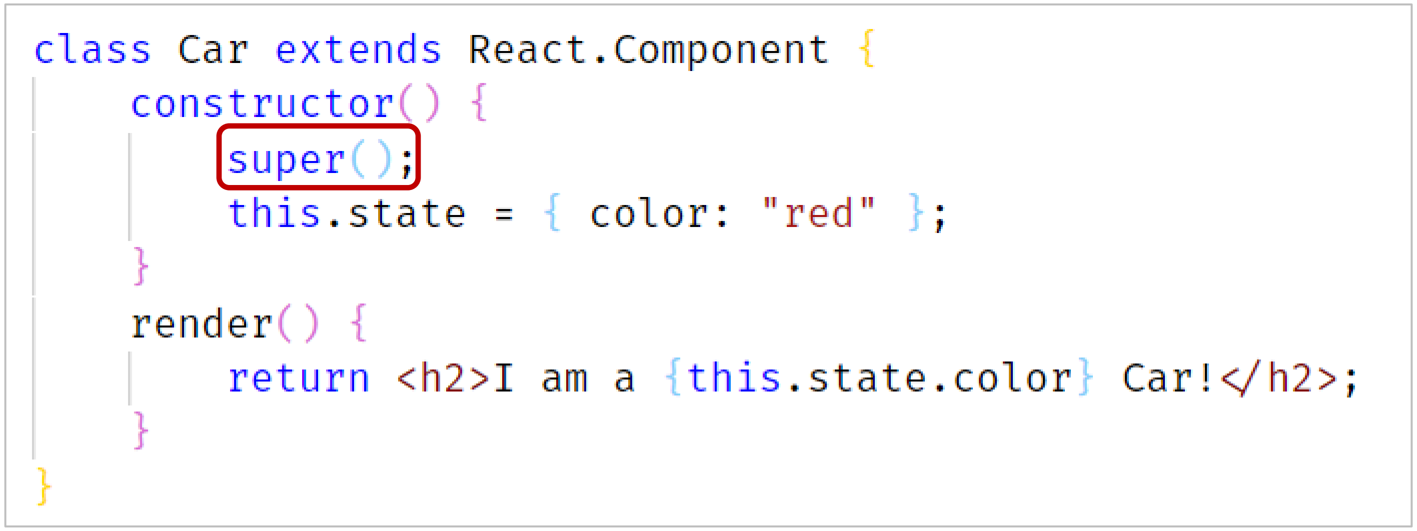
** Component name MUST start with an uppercase letter*

Component Constructor

- Called when the component gets initiated
  - initiate the **component's properties**
  - inherit parent component super()

- In React, component's properties should be kept in an object called **state**

  e.g. add color property & use it in render()

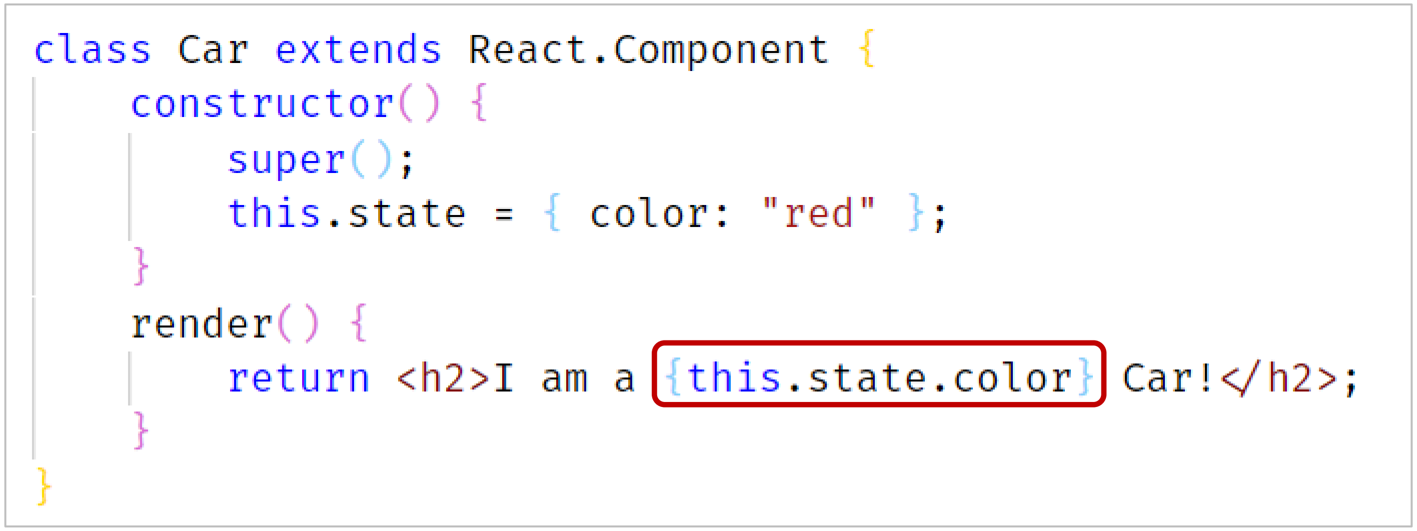


```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: "red" };
    }
    render() {
        return <h2>I am a {this.state.color} Car!</h2>;
    }
}
```

Using the state object

- Refer to the **state** object anywhere in the component by using syntax:

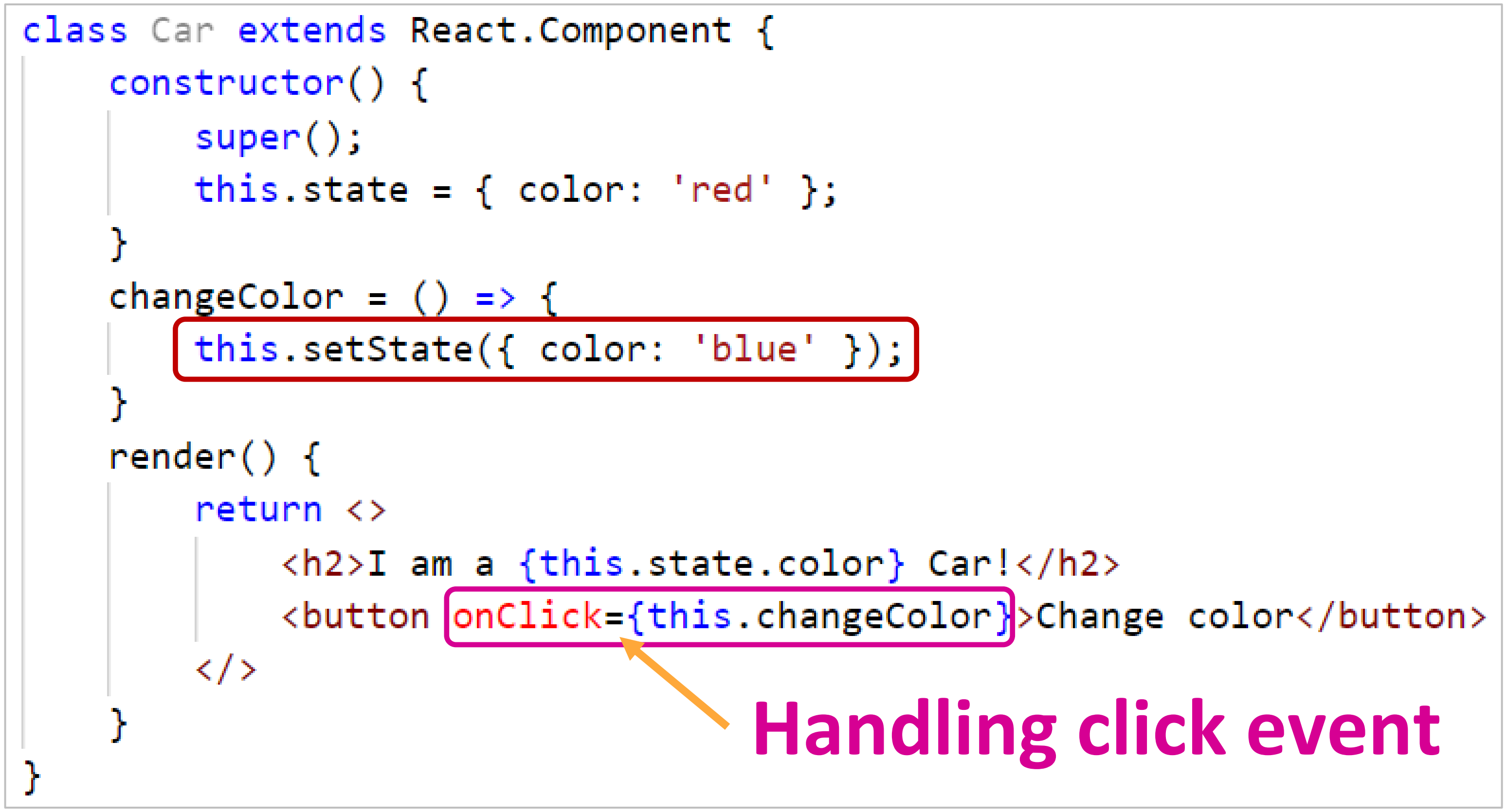
```
this.state.propertyname
```



```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: "red" };
    }
    render() {
        return <h2>I am a {this.state.color} Car!</h2>;
    }
}
```

Changing the state object

- Use `this.setState()` method.



```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: 'red' };
    }
    changeColor = () => {
        this.setState({ color: 'blue' });
    }
    render() {
        return <>
            <h2>I am a {this.state.color} Car!</h2>
            <button onClick={this.changeColor}>Change color</button>
        </>
    }
}
```

- When state object changes → the component re-renders.

Important note on State

Always use the `setState()` method
to change the state object.

- it will ensure that the component knows its been updated

→calls the `render()` method

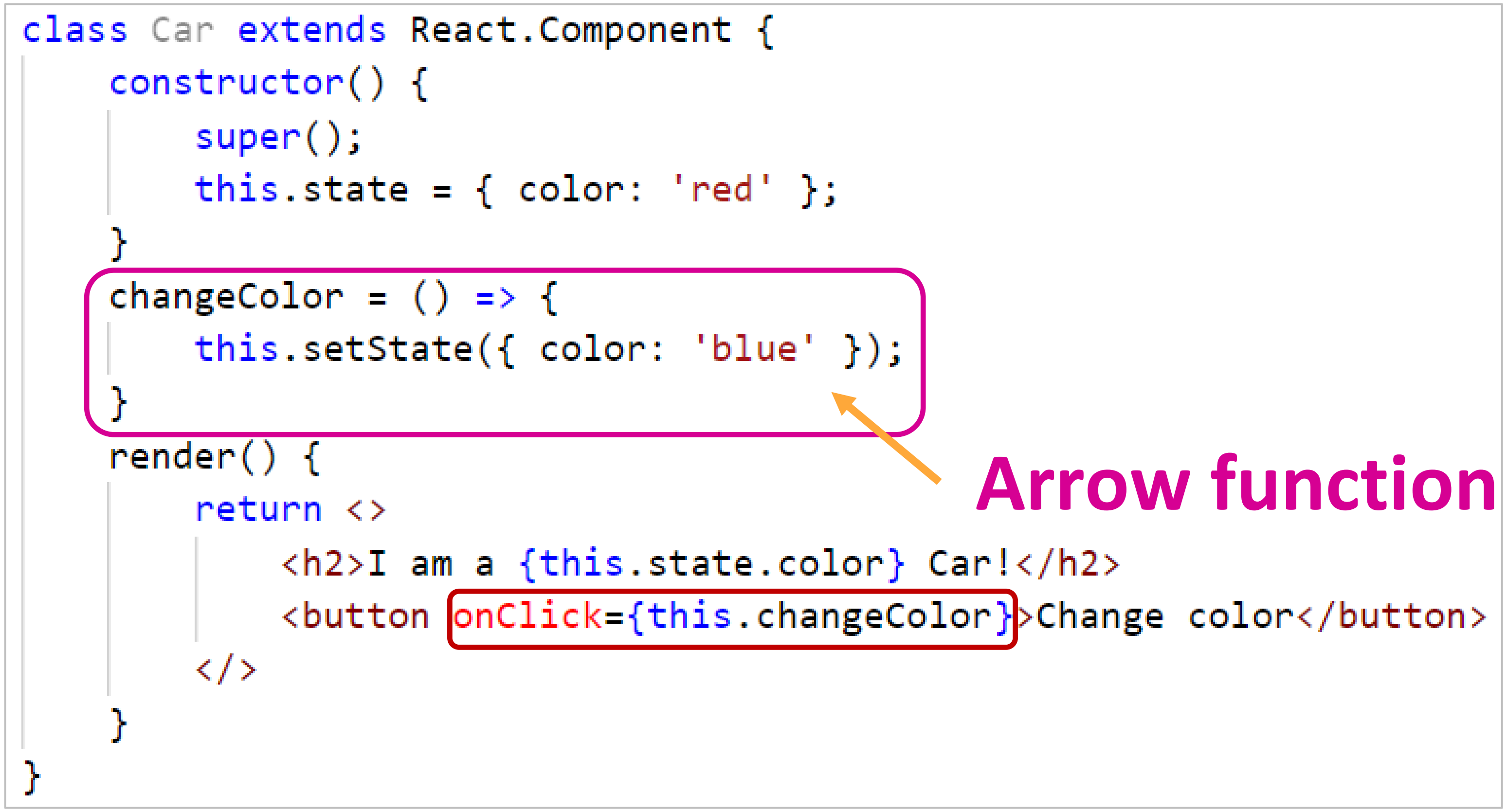
→(and all the **other lifecycle methods**) ???

Today

Handling click event

addEventListener? → No, React makes it easier

- Attribute: onClick

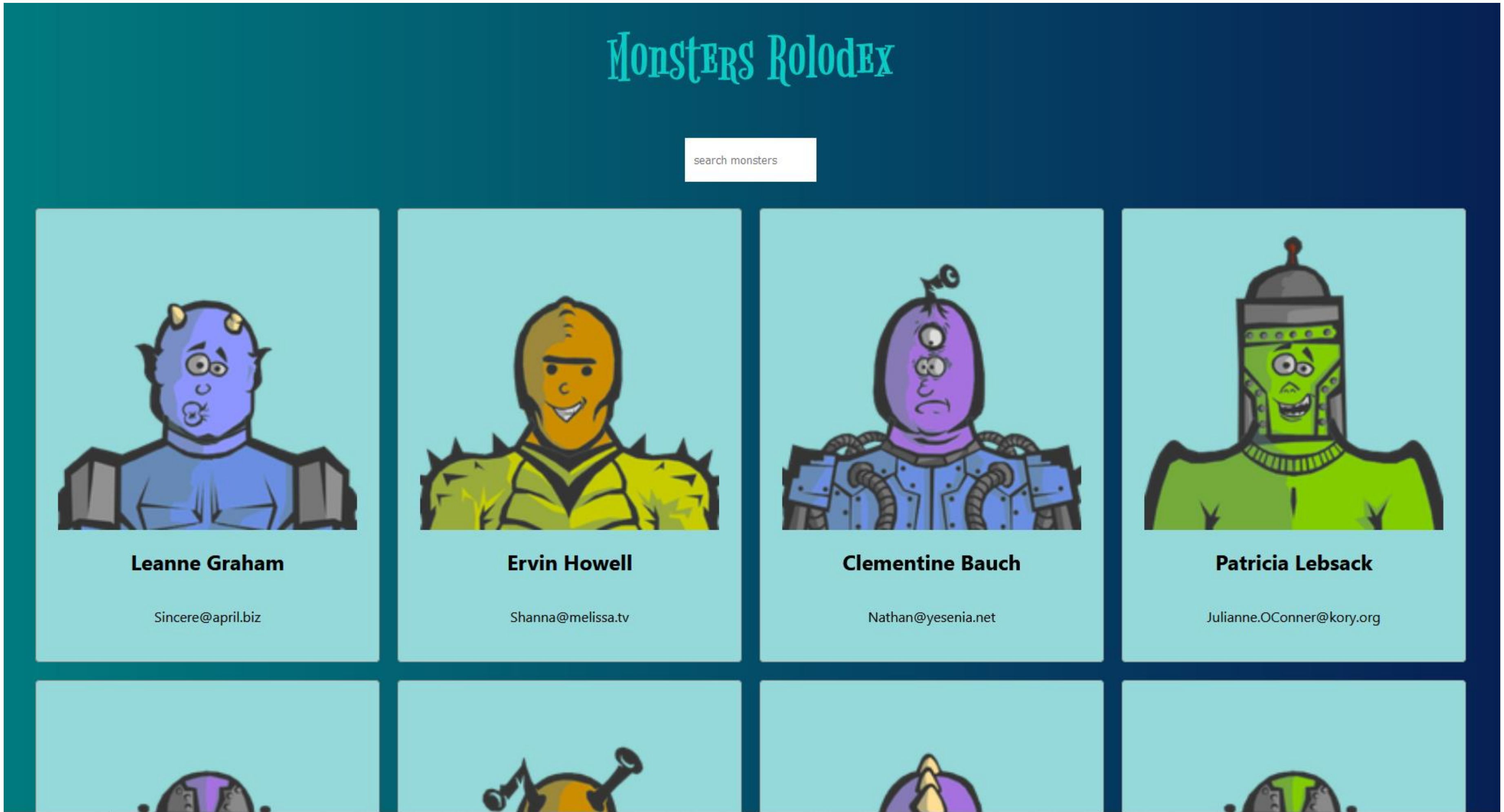


```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: 'red' };
    }
    changeColor = () => {
        this.setState({ color: 'blue' });
    }
    render() {
        return <>
            <h2>I am a {this.state.color} Car!</h2>
            <button onClick={this.changeColor}>Change color</button>
        </>
    }
}
```

Arrow function

Example: monsters

`[code demo]`



Leanne Graham
Sincere@april.biz

Ervin Howell
Shanna@melissa.tv

Clementine Bauch
Nathan@yesenia.net

Patricia Lebsack
Julianne.OConner@kory.org

Fonts

public/index.html

```
<link
  href="https://fonts.googleapis.com/css?family=Bigelow+Rules"
  rel="stylesheet" />
```

CSS in React

- Write your CSS styling in a separate `.css` file

→ Import it in your application

App.css

```css
body {
  background-color: #282c34;
  color: white;
  padding: 40px;
  font-family: Arial;
  text-align: center;
}
```

index.js

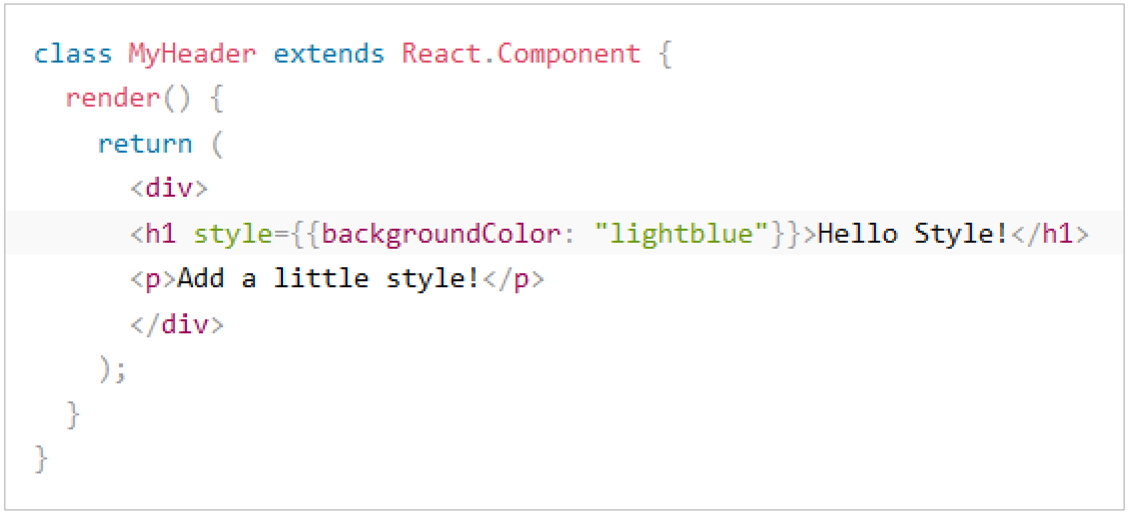
```javascript
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';

class MyHeader extends React.Component {
  render() {
    return (
      <div>
      <h1>Hello Style!</h1>
      <p>Add a little style!.</p>
      </div>
    );
  }
}
```

CSS in React

Some notes:

- class → className
- for → htmlFor

Inline style – **camelCase** property names:



```
class MyHeader extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
      <p>Add a little style!</p>
      </div>
    );
  }
}
```

Dynamic content

- Store JSX in variables

```
render() {
    const innerDiv = <div className='inner'>Inner</div>;

    return <div className='outter'>
      {innerDiv}
    </div>;
  }
```

Dynamic content

- Display collection of data in JSX
  - **IMPORTANT**: Attribute: key

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

```
render() {
  const cards = [];

  for (const monster in this.state.monsters) {
    const card = <div
      className='card-container'
      key={monster.email} >

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

  return <div class="card-list">
    {cards}
  </div>;
}
```

Selected topic: map() function

```
const numbers = [4, 9, 16, 25];
const newArr = numbers.map(Math.sqrt);
```

- map(): creates a new array from calling a function for every array element.

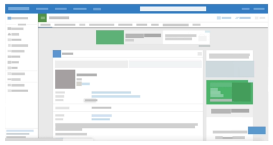
```
class App extends React.Component {
  constructor() {
    super();
    this.state = [
      {
        name: 'CongNV',
        email: 'congnv@hanu.edu.vn'
      },
      {
        name: 'CamNH',
        email: 'camnh@hanu.edu.vn'
      }
    ];
  }
```

```
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>;
}
```

Practice: Architecting our app

Recall: The job of a React Developer

1. Decide on Components



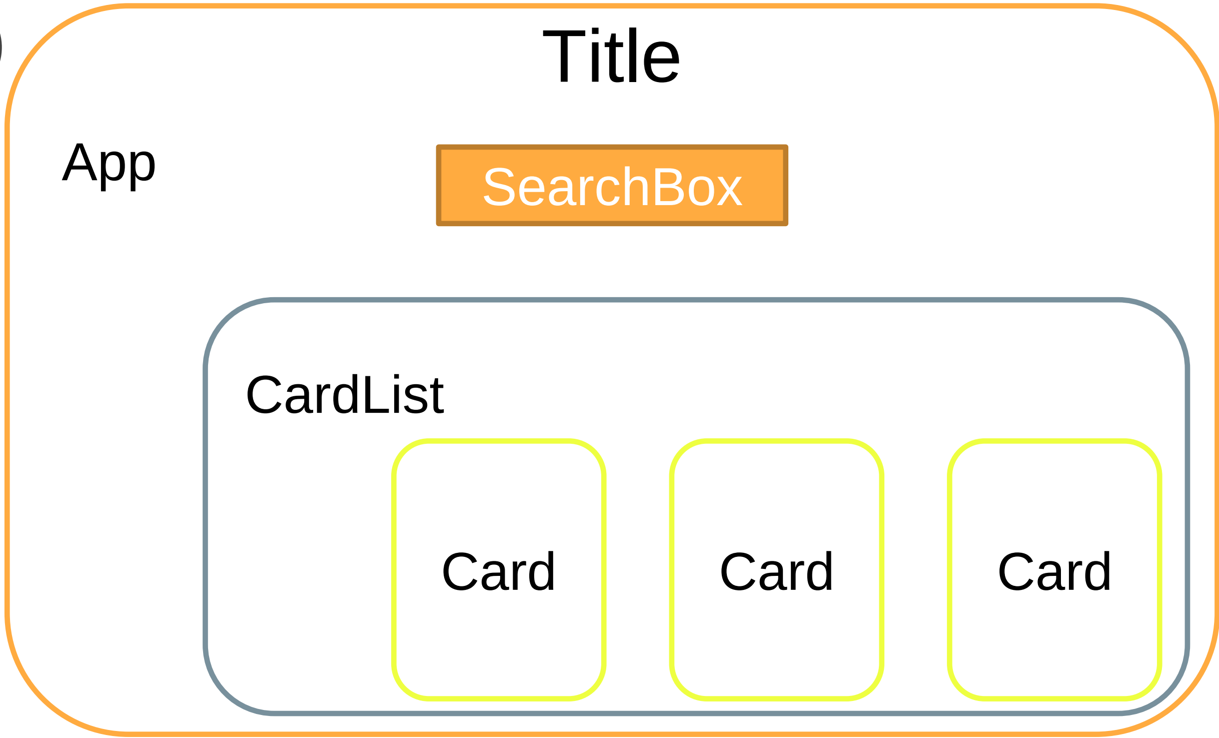
2. Decide the State and where it lives

3. What changes when state changes

Example: monsters

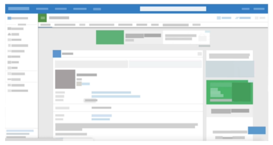
- **App**: the app
- **CardList**: loop monsters to display as cards
- **Card**: display a monster

- **SearchBox** (later)



Recall: The job of a React Developer

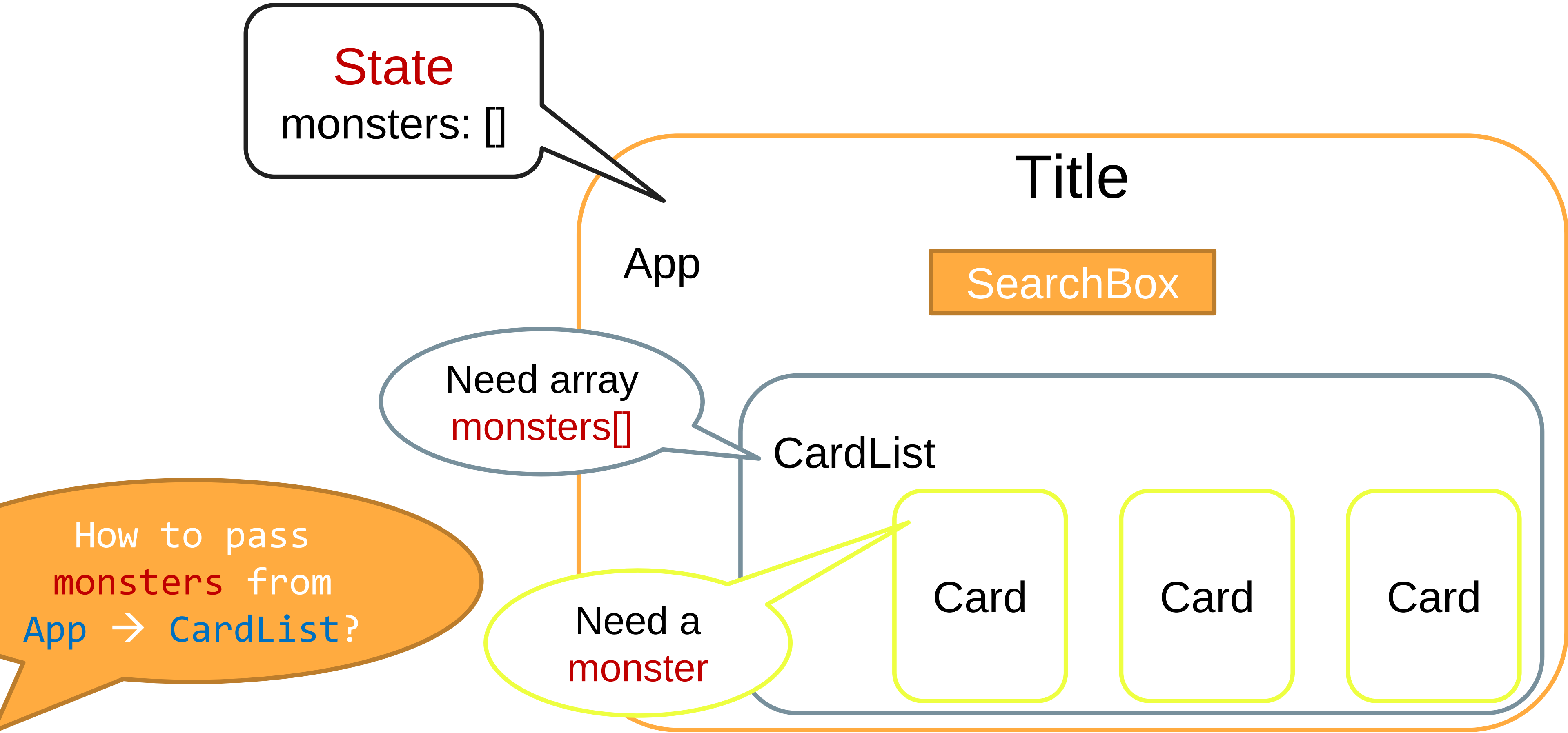
1. Decide on Components



2. Decide the State and where it lives

3. What changes when state changes

Example: monsters



How to pass data into component?

Props

Props

- Props = **function arguments**

```
function Car(props) {
    return <h2>I am a {props.brand}!</h2>;
}
```

- Passed to components via HTML attributes

  e.g. add a *brand* attribute to Car component

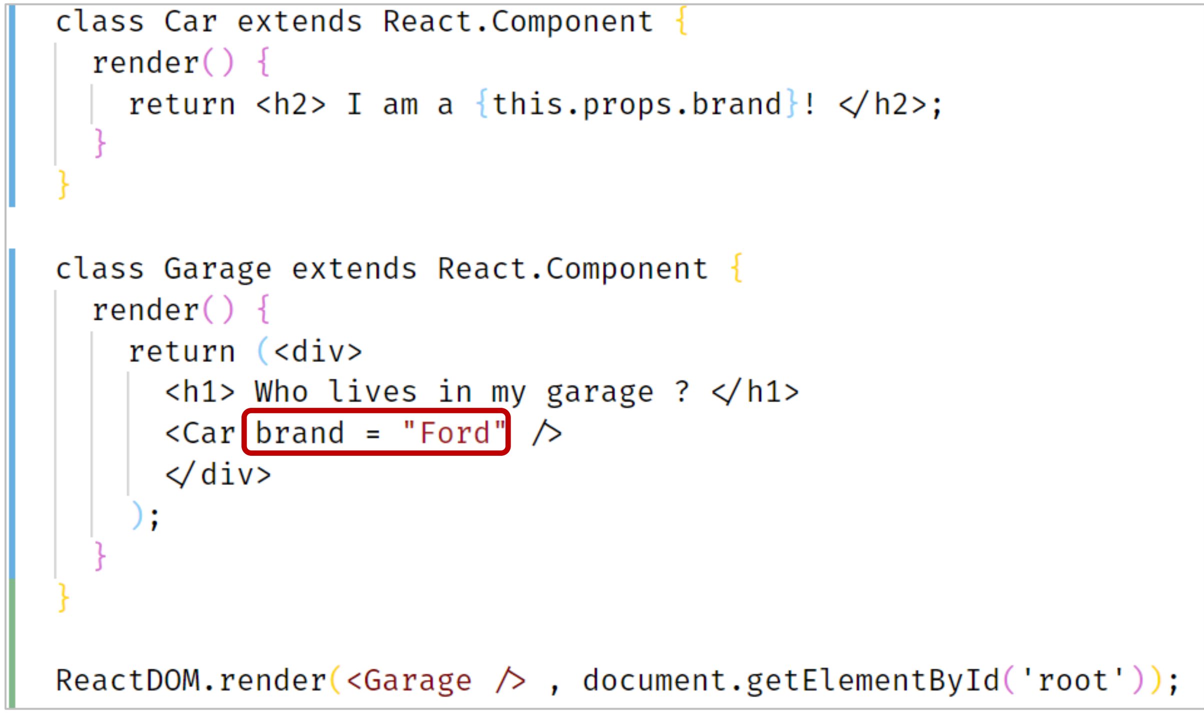
```
  const myelement = <Car brand = "Ford" /> ;
```

- The arguments are received as **props** object

```
  class Car extends React.Component {
    render() {
      return <h2> I am a {
        this.props.brand
      }! </h2>;
    }
  }
```

Passing Data

e.g.
Send "brand“
from **Garage**
to **Car**

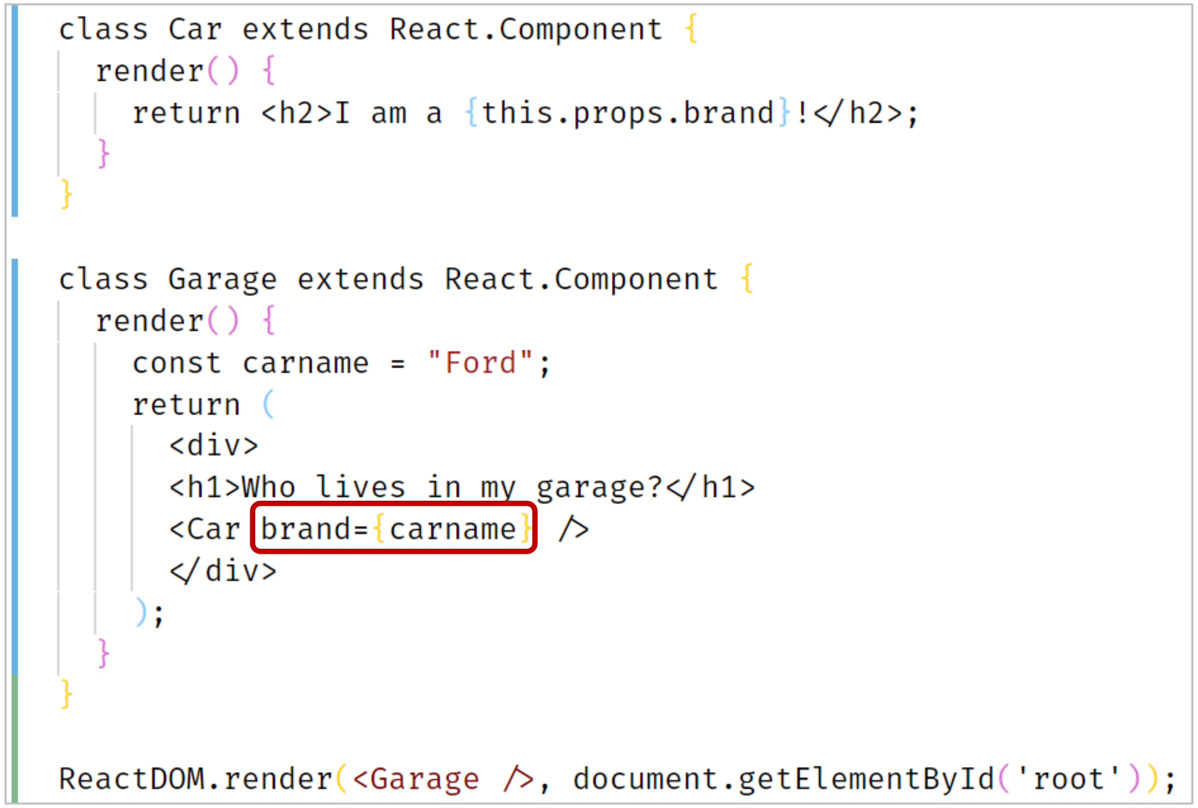


```
class Car extends React.Component {
  render() {
    return <h2> I am a {this.props.brand}! </h2>;
  }
}

class Garage extends React.Component {
  render() {
    return (<div>
      <h1> Who lives in my garage ? </h1>
      <Car brand = "Ford" />
      </div>
    );
  }
}

ReactDOM.render(<Garage /> , document.getElementById('root'));
```

Passing Data

Pass a variable – NOT a string

→ Put the variable name inside {}



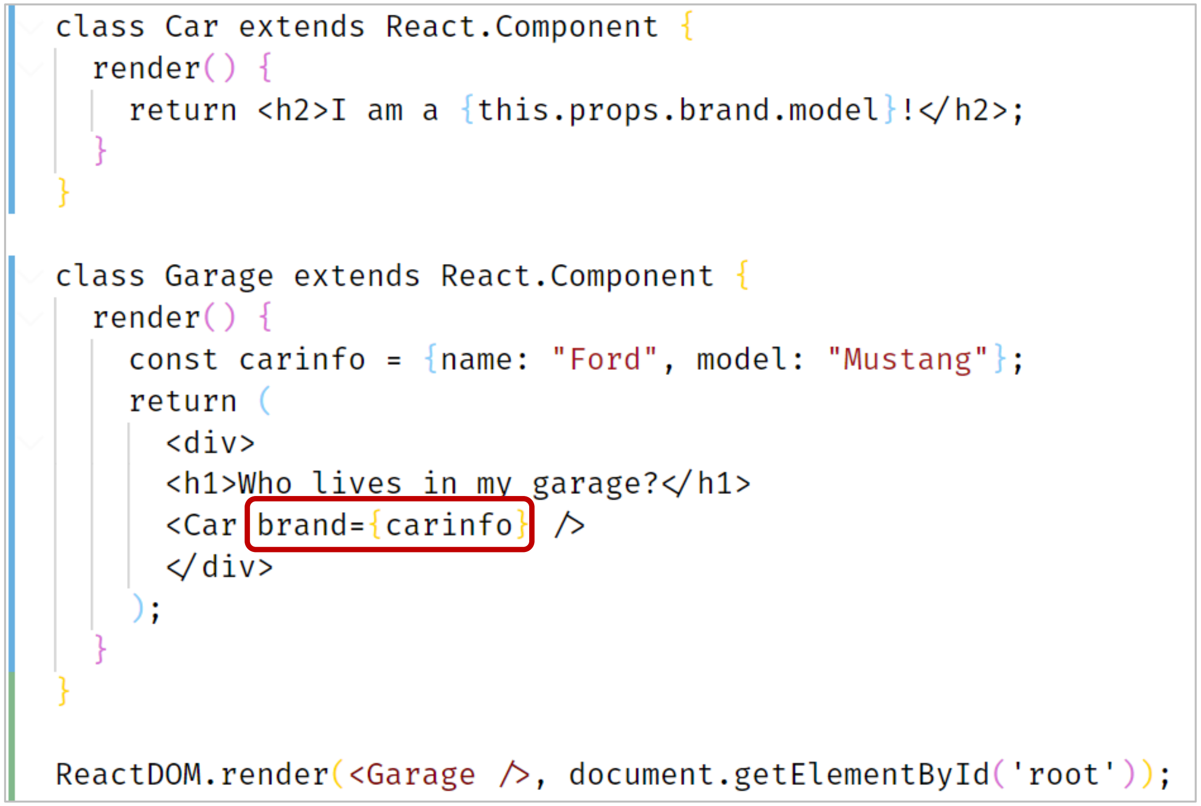
```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand}!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    const carname = "Ford";
    return (
      <div>
      <h1>Who lives in my garage?</h1>
      <Car brand={carname} />
      </div>
    );
  }
}

ReactDOM.render(<Garage />, document.getElementById('root'));
```

Passing Data

Pass a variable – NOT a string

OR an object

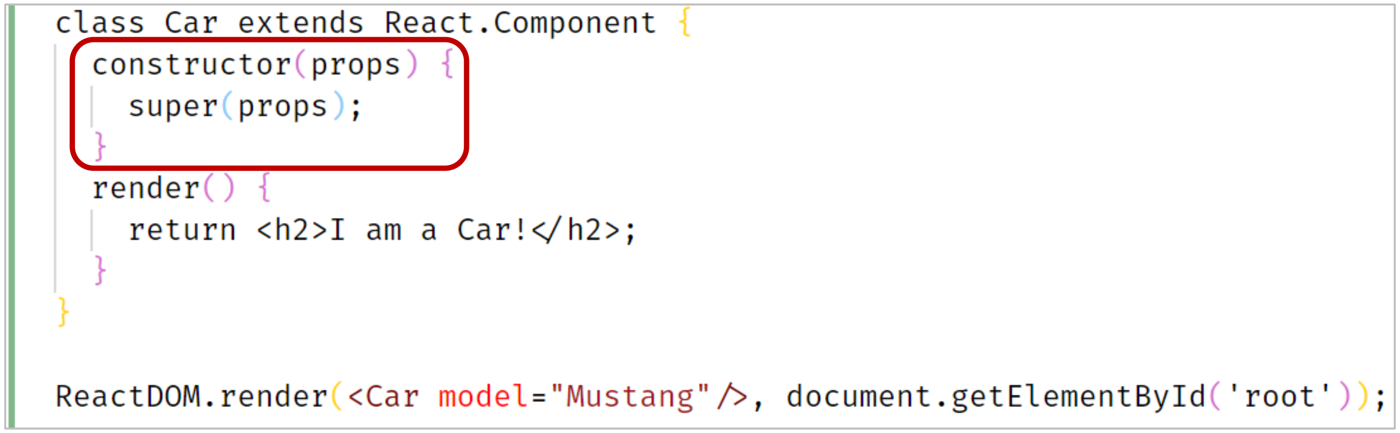
→ Put the variable name inside {}



```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand.model}!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    const carinfo = {name: "Ford", model: "Mustang"};
    return (
      <div>
      <h1>Who lives in my garage?</h1>
      <Car brand={carinfo} />
      </div>
    );
  }
}

ReactDOM.render(<Garage />, document.getElementById('root'));
```

Props in the constructor

- If constructor,
  - the props should **always** be passed to the constructor via the `super()` method



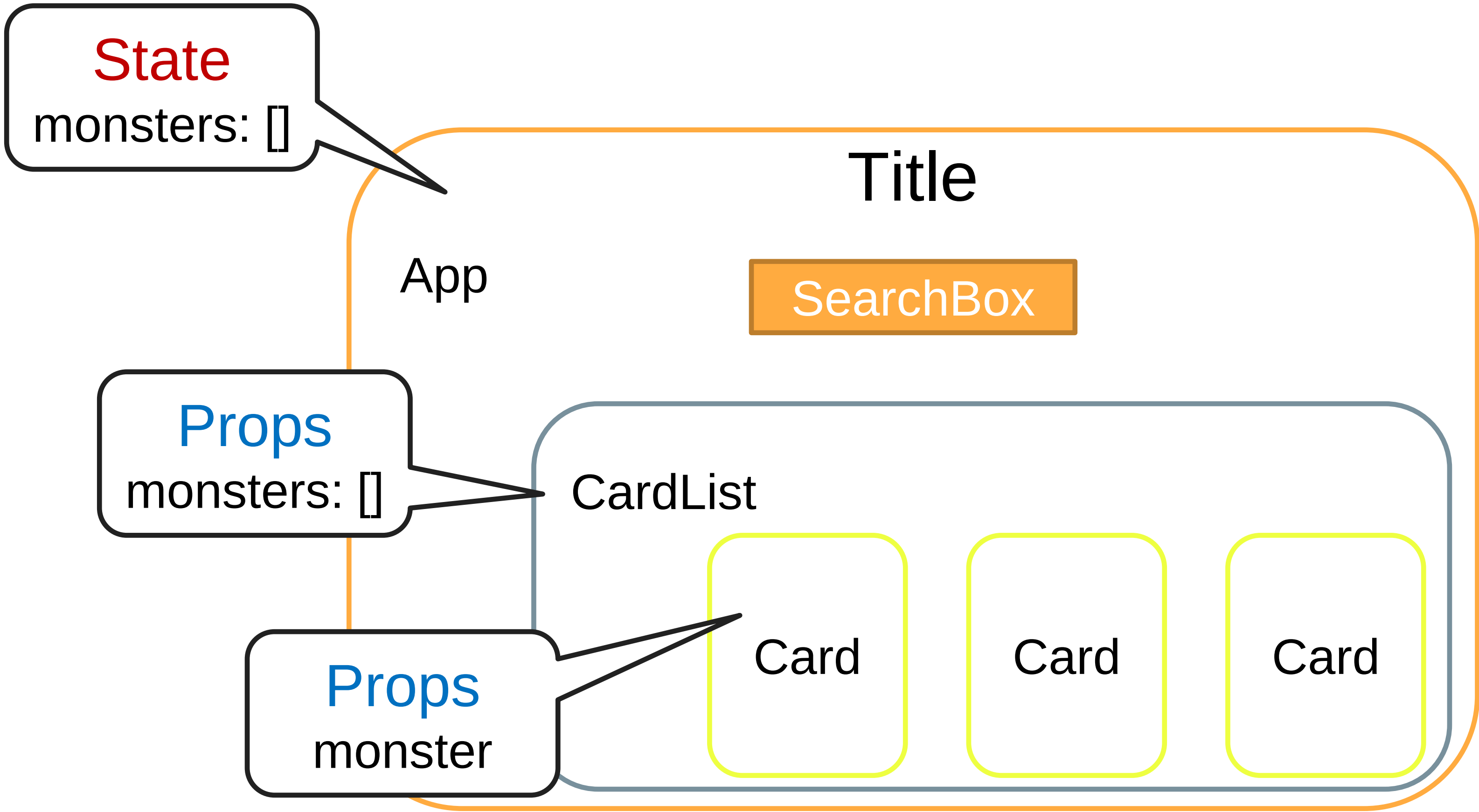
```
class Car extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <h2>I am a Car!</h2>;
  }
}

ReactDOM.render(<Car model="Mustang"/>, document.getElementById('root'));
```

Important note on Props

React Props are **read-only**!

You will get an error if you try to change their value.

Example: monsters



Example: monsters

Fetching data:

- Data: https://jsonplaceholder.typicode.com/users
- Image: https://robohash.org/1?set=set2&size=180x180

Example: monsters

[code demo]

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

    this.state = {
      monsters: [],
      searchField: ''
    };
  }

  componentDidMount() {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(users => this.setState({ monsters: users }));
  }
```

`componentDidMount()?`

Component Lifecycle

React component lifecycle

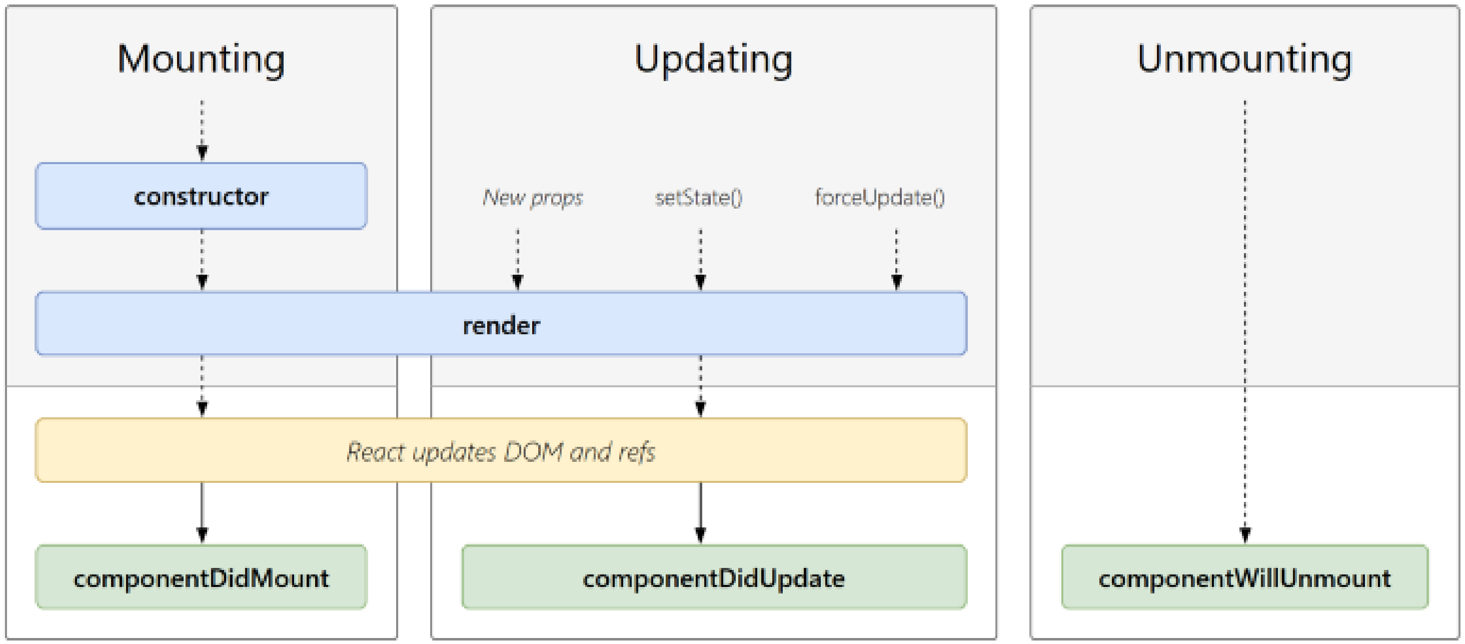
Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

The three phases are:

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

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

React component lifecycle



More?
Next week!