State & Lifecycle

Session 03

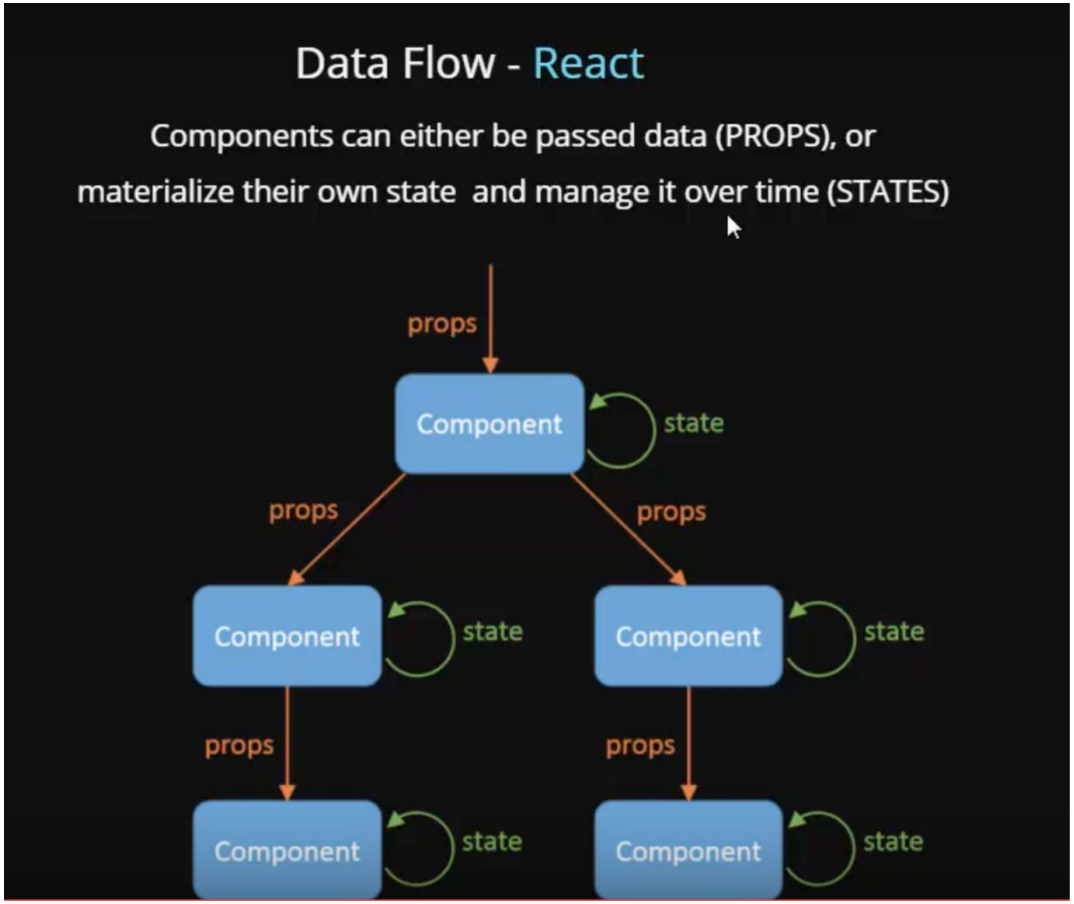
State

- State được tạo ra và quản lý bởi component hiện tại
- State được dùng cho những dữ liệu khả năng sẽ thay đổi

Props & State

Truyền dữ liệu:

- Từ cha → con: Props
- Giữa component ngang hàng: để state trên component cha, rồi từ cha → con
- Giữa các trang khác nhau: Redux (Slide 09)


Data Flow - React
Components can either be passed data (PROPS), or
materialize their own state and manage it over time (STATES)
props
Component
state
props
props
Component
state
Component
state
props
props
Component
state
Component
state

Component life cycle

Với component trong ReactJS, life cycle gồm 3 giai đoạn:

- Được tạo ra (Mounting)
- Qua nhiều thay đổi (Updating)
- Và bị hủy bỏ (Unmounting)

“Render Phase”

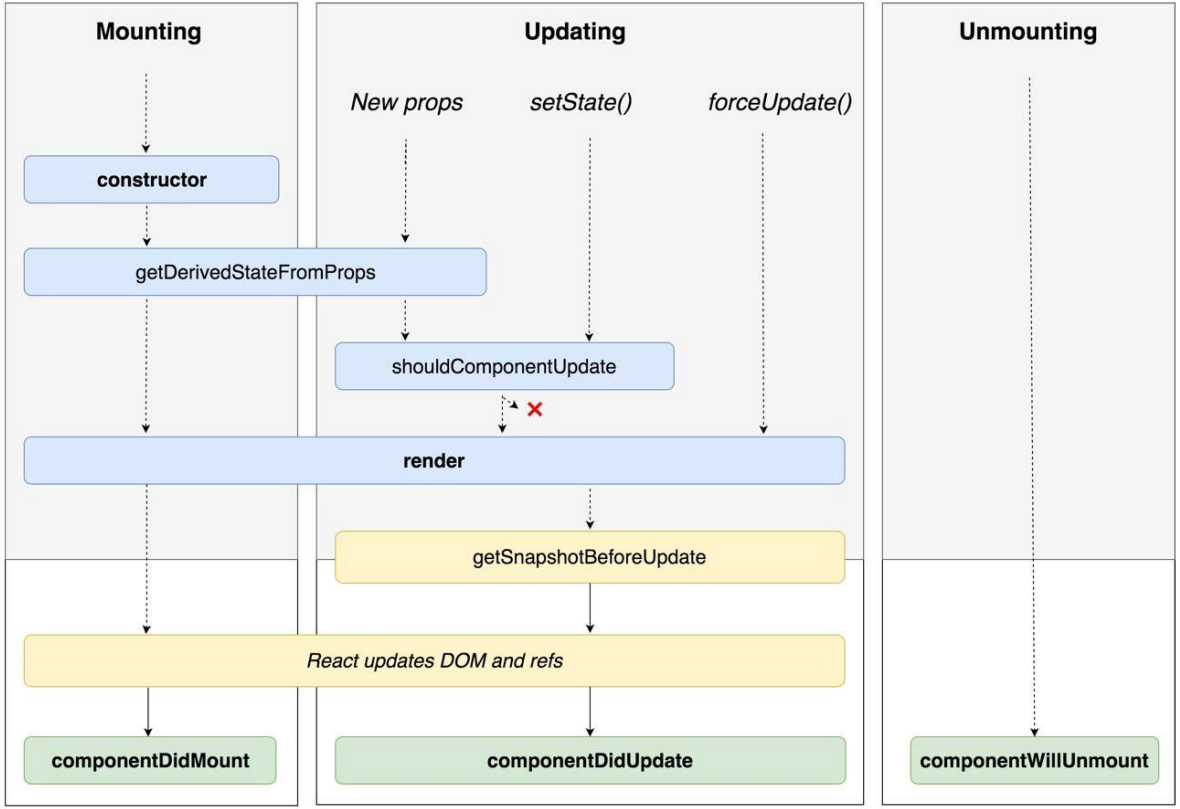
Pure and has no side effects. May be paused, aborted or restarted by React.

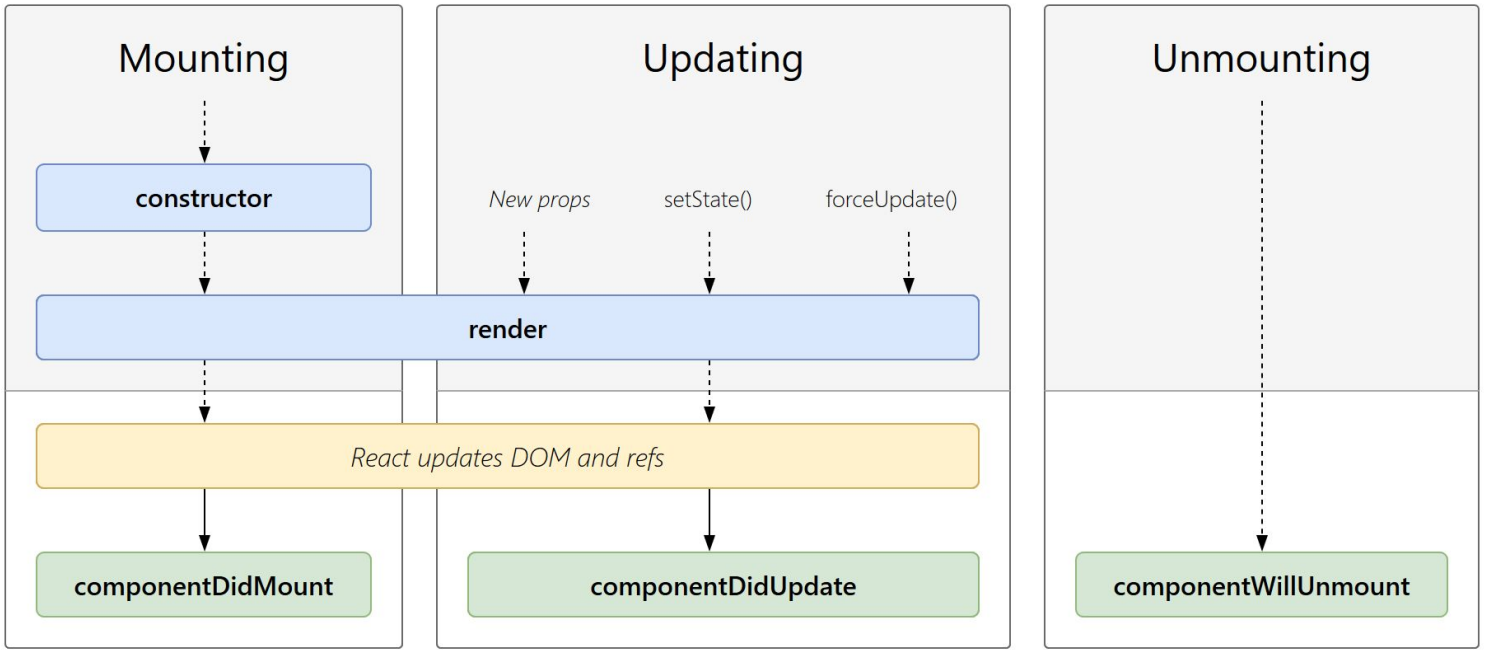
“Pre-Commit Phase”

Can read the DOM.

“Commit Phase”

Can work with DOM, run side effects, schedule updates.




Mounting
constructor
Updating
New props
setState()
forceUpdate()
render
Unmounting
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

Nên	Không Nên	
constructor		Nhớ có super(props) Khai báo state Định nghĩa properties của component
componentDidMount		Khởi tạo dữ liệu cho component: gọi API, biến đổi dữ liệu, cập nhật state Gửi tracking page view (GA, FacebookPixel)
componentWillUnmount		Clear timeout hoặc interval nếu có dùng Reset dữ liệu trên Redux nếu cần
componentDidUpdate		Hạn chế dùng ADVANCED Chỉ dùng nếu muốn handle update component khi click nút back mà trên URL có query params
	componentWillMount	Cần refactor lại code nếu có đang dùng.
	componentWillReceiveProps	
PureComponent		Vì có shallow comparision trong hàm shouldComponentUpdate. https://stackoverflow.com/questions/41340697/react-component-vs-react-purecomponent/53740921

Handling events

Handling events với React Element cũng tương tự như handling event trên DOM Element

Khác:

- Tên của React Events được đặt theo kiểu camelCase chứ không phải lowercase
- Truyền JSX vào một function để handle event chứ không phải là một string

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

Handling events

Bạn không thể gọi *return false* để dừng hành động mặc định (prevent default) trong React. Bạn phải gọi *preventDefault*

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

```
<a href="#" onclick="console.log('The link was clicked.'); return false">
  Click me
</a>
```

Handling events

- *e* là tổng hợp tất cả event. React js xác định các event này theo W3C spec. Khi sử dụng React, bạn không cần gọi function *addEventListener* để lắng nghe sự kiện vào DOM khi nó tượng tạo mới. Thay vào đó, chỉ cần khai báo một sự kiện lắng nghe khi nó được hiển thị lần đầu. Ví dụ, với component *Togged*, sẽ cho phép người dùng click vào button chuyển trạng thái từ ON -> OFF
- Lưu ý: Cẩn phải thận trọng với việc sử dụng *this* trong việc JSX callbacks. Trong Javascript, các method của class không bị ràng buộc bởi mặc định. Nếu bạn quên không gọi *this.handleClick* mà gọi thằng *onClick* khi đó *this* sẽ bị lỗi *undefined* khi mà function được gọi.

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({
      isToggleOn: !prevState.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

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

Nếu bạn thấy gọi binds là bất tiện. Có 2 cách giúp bạn cải thiện việc này. Sử dụng field class để gọi và dùng arow function

```
class LoggingButton extends React.Component {
  // This syntax ensures `this` is bound within handleClick.
  // Warning: this is *experimental* syntax.
  handleClick = () => {
    console.log('this is:', this);
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
    );
  }
}
```

```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('this is:', this);
  }

  render() {
    // This syntax ensures `this` is bound within handleClick
    return (
      <button onClick={(e) => this.handleClick(e)}>
        Click me
      </button>
    );
  }
}
```

Để truyền tham số vào event handlers, có thể dùng theo 1 trong số cách sau:

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

Trong cả hai trường hợp, tham số e đại diện cho React Event sẽ được truyền như là tham số thứ 2 sau id.

Conditional Rendering

Trong React bạn có thể tạo các component riêng biệt chứa những hành động bạn cần. Conditional Rendering hoạt động tương tự với Conditional trong Javascript. Sử dụng javascript operator như là if tạo các phần tử đại diện cho trạng thái hiện tại.

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

Conditional Rendering

Chúng ta sẽ tạo một component Greeting hiển thị một trong 2 thành phần bên trên tùy thuộc vào việc người dùng có login vào hệ thống hay không:

Khai báo một biến và sử dụng if là cách tốt nhất để render một thành phần có điều kiện

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```