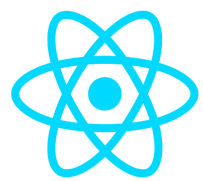
Why React ?

Why React Hooks ??

wait... why am I here ??? 

By `/^(Ry|Bri)an$/gm`

React Highlights



- vDOM
- Flux (one way data flow)
- Component Composition/Reuse

- `<JSX />`...

next =>

Class or Functional

Class component

Component

```
class MyComponent extends React.PureCompon
    constructor() {
        this.state = {
            value: ''
        };
        onClick = this.onClick.bind(this)
    }

    componentDidMount() {
        //window.document.title = `${this.
        document.title = "WHY React"
    }

    componentDidUpdate() {
        //window.document.title = `${this.
    }

    onClick(e) {
        this.setState({
            value: e.target.value
        })
    }

    render() {
        return (
            <>
                <p>{this.state.value} Comp
                <input
                    onChange={e => this.on
                    value={this.state.valu
            </>
        );
    }
}
```

Functional/ Stateless Components

Functional Component

```
() => {
    const text = "Functional Co
    return <p>{text}</p>
}
```

Class Comp.

- Access to states/ lifecyle methods
- ... but it's wordy and not webpack friendly

Functional

- Only stateless controls, `props` to `JSX`

... Functional

Stateless ???

What if... Don't look at the code yet lol

Component

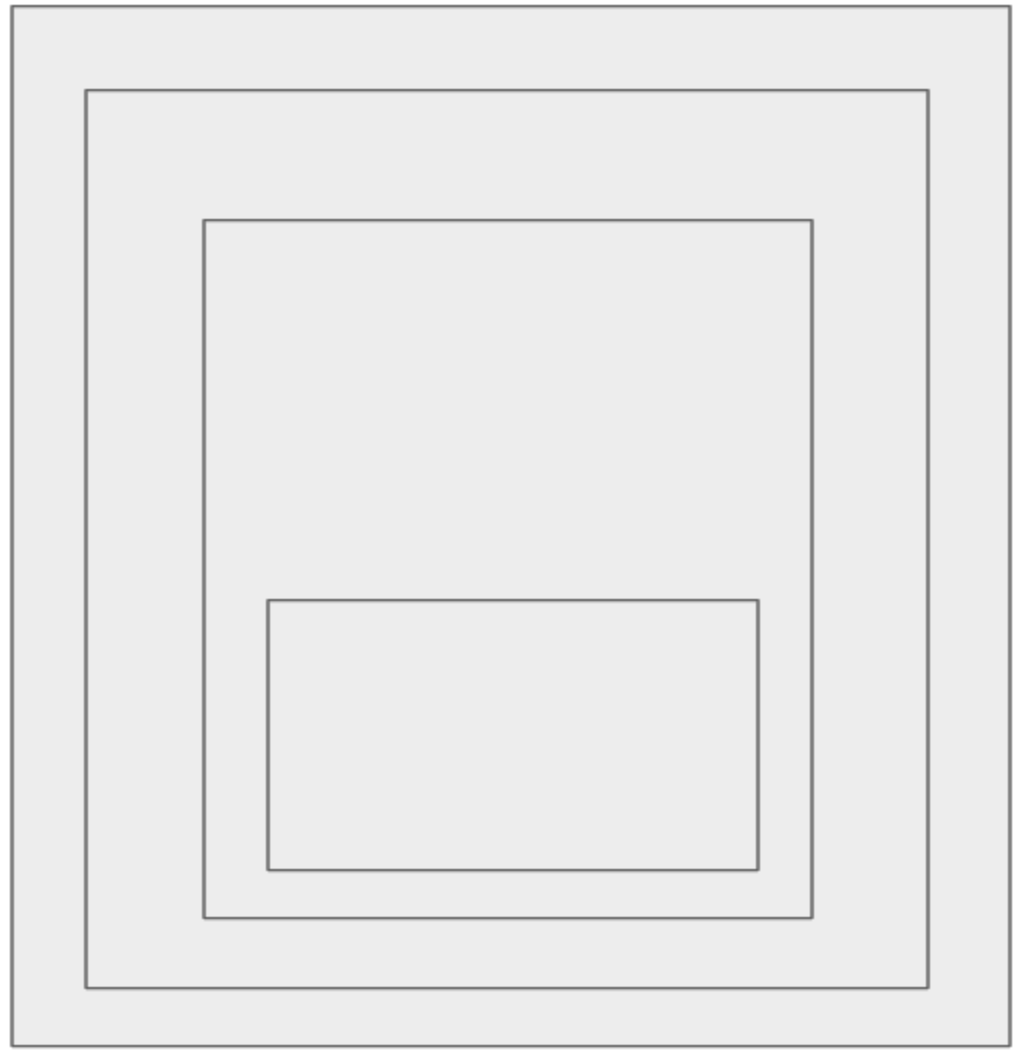
```
() => {
    const [text, setText] = React.useState

    React.useEffect(() => {
        //window.document.title = `${text}
        window.document.title = "WHY React
    });

    return (
        <>
            <p>{text} Component</p>
            <input
                onChange={e => setText(e.t
                value={text}/>
        </>
    )
}
```

Wait a second...

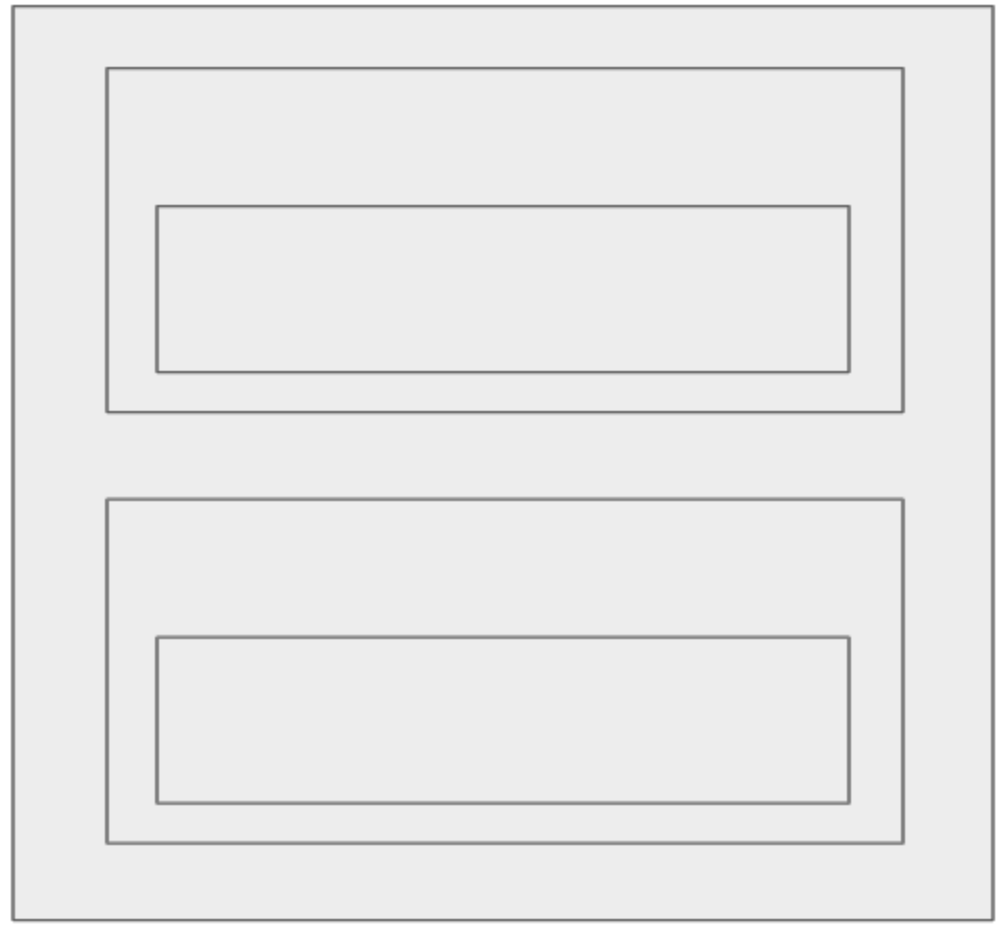
... we've more `<code/>`

```
dComponent = withStyles(({ color }) => ({}))(MyComponent);14. 15. 16.     const App = withRouter(17.
```

HoC



renderProps



```
ousSlide} 8.                icon="sapIcon-previous" 9.          />10.     )}11.     renderCen
```

Wrapper Hell

```
▼ <Unknown>
  ▼ <t debug={false} errorMessage="">
    ▼ <o>
      ▼ <t>
        ▼ <t>
          ▼ <Router>
            ▼ <RouterContext>
              ▼ <Apollo(Connect(Apollo(n)))>
                ▼ <t fetchPolicy="network-only" errorPolicy="ignore" ssr={false} displayName="Apollo(Connect(Apollo(n)))"
                  skip={false} warnUnhandledError={true}>
                  ▼ <Connect(Apollo(n)) authLoading={false} isAuthenticated={2539615}>
                    ▼ <Apollo(n) authLoading={false} isAuthenticated={2539615}>
                      ▼ <t errorPolicy="ignore" ssr={false} displayName="Apollo(n)" skip={false} warnUnhandledError={true}>
                        ▼ <n authLoading={false} isAuthenticated={2539615} userLoading={false}>
                          ▼ <Connect(Apollo(t)) authLoading={false} isAuthenticated={2539615} userLoading={false}>
                            ▶ <Apollo(t) authLoading={false} isAuthenticated={2539615} userLoading={false} isMobile={false
                              } isOnline={true} lang="id" popUp={false} searchModalOpen={false} sessionId={2539615} xdevice
                              ="">…</Apollo(t)> == $r
                          </Connect(Apollo(t))>
                        </n>
                      </t>
                    </Apollo(n)>
                  </Connect(Apollo(n))>
                </t>
              </Apollo(Connect(Apollo(n)))>
            </RouterContext>
          </Router>
        </t>
      </t>
    </o>
  </t>
</Unknown>
```

Solution lol

...but there is hope...

useState

useEffect

useContext

useReducer

...

useYourOwn

Questions?

```
ntDidUpdate() {        //window.document.title = `${this.state.value} Component`    }    onClick(e) {        t
```

```
n we'll call "count"  const [title, setTitle] = useState("");  return (    <>      <p>{tit
```

```
ls.handleStatusChange    );  }  componentWillUnmount() {    ChatAPI.unsubscribeFromFriendStatus(      this.props.friend.id,      this.handleSta
```

```
ops.friend.id, handleStatusChange);    // Specify how to clean up after this effect:    r
```

Basic Rules

Only Call Hooks at the Top Level

Don't call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function. By following this rule, you ensure that Hooks are called in the same order each time a component renders. That's what allows React to correctly preserve the state of Hooks between multiple `useState` and `useEffect` calls.

Only Call Hooks from React Functions

Don't call Hooks from regular JavaScript functions. Instead, you can:

- ✅ Call Hooks from React function components.
- ✅ Call Hooks from custom Hooks (we'll learn about them on the next page).

By following this rule, you ensure that all stateful logic in a component is clearly visible from its source code.

```
eturn (    <ThemeContext.Consumer>      {theme => (        <section className={theme}>
```

```
values, or locale etc.function Display() {  const theme = useContext(ThemeContext);  cons
```

```
e(status.isOnline);  }  useEffect(() => {    ChatAPI.subscribeToFriendStatus(friendID, ha
```

Usage

```
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

```
function FriendListItem(props) {
  const isOnline = useFriendStatus(props.friend.id);

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

Thank You! Any Questions?