React: Crash Course

1. Thinking in React
2. React
3. State Management
4. Next Steps
5. Try it Yourself

Thinking in React

1. **Cohesion**: Increase cohesion, decrease coupling.
2. **Data Flow**: All data flows in one direction.

Thinking in React

Cohesion

- Features first, types second.
- Easy to package.

Thinking in React

Cohesion

```
src/
|__ Form/
    |__ Input/
    |   |__ Input.js
    |   |__ Input.styles.css
    |   |__ index.js
    |__ Button/
    |__ Form.js
    |__ index.js
```

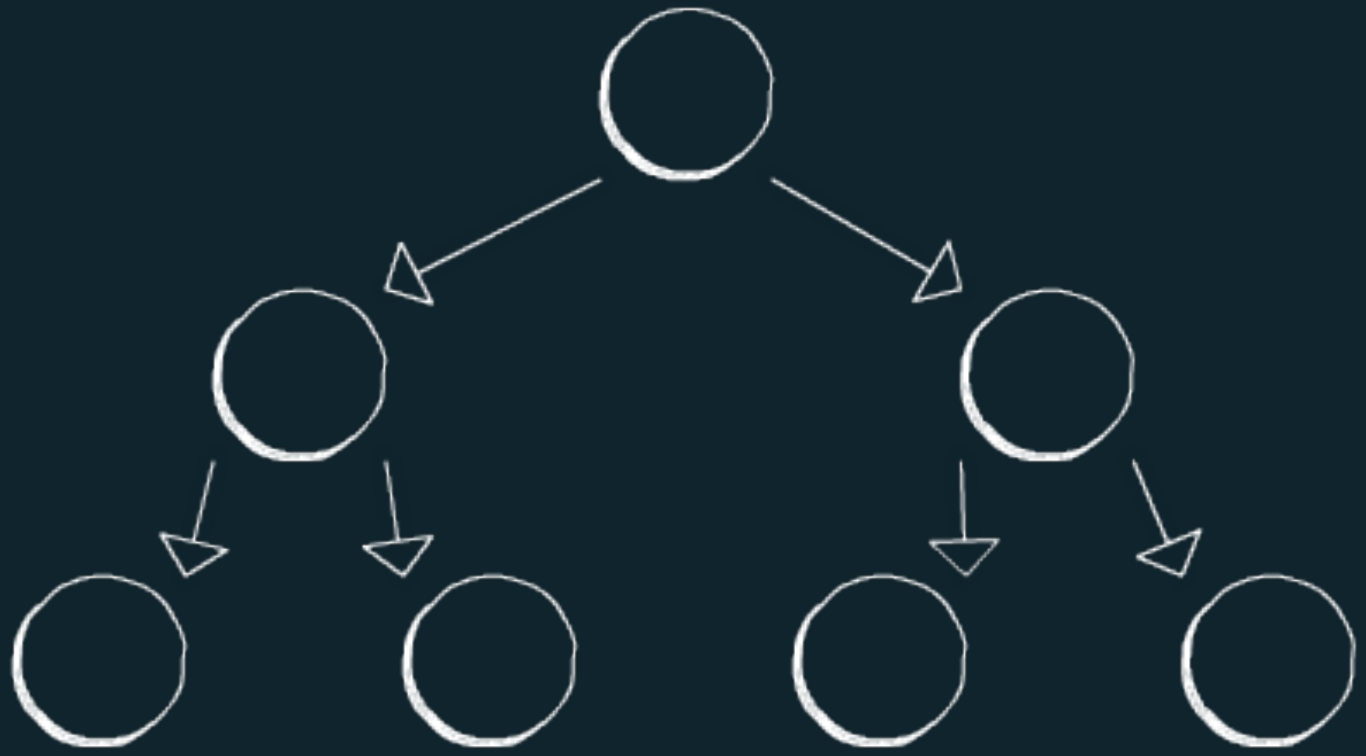
Thinking in React

Data Flow

- All data flows downwards.
- There is no* way to pass information upwards.

Thinking in React

Data Flow



Component Tree

React

React

Cohesion

```
// Button.js
function Button(props) {
  return (
    <button onClick={handleOnClick}>{props.callToAction}</button>
  )
}

function handleOnClick() {
  console.log('clicked!');
}
```

Increase cohesion between related HTML and JS.

React

Cohesion

```
import btn from "./Button.styles";

function Button(props) {
  return (
    <button className={btn.primary}>{props.callToAction}</button>
  )
}
```

React

Data Flow

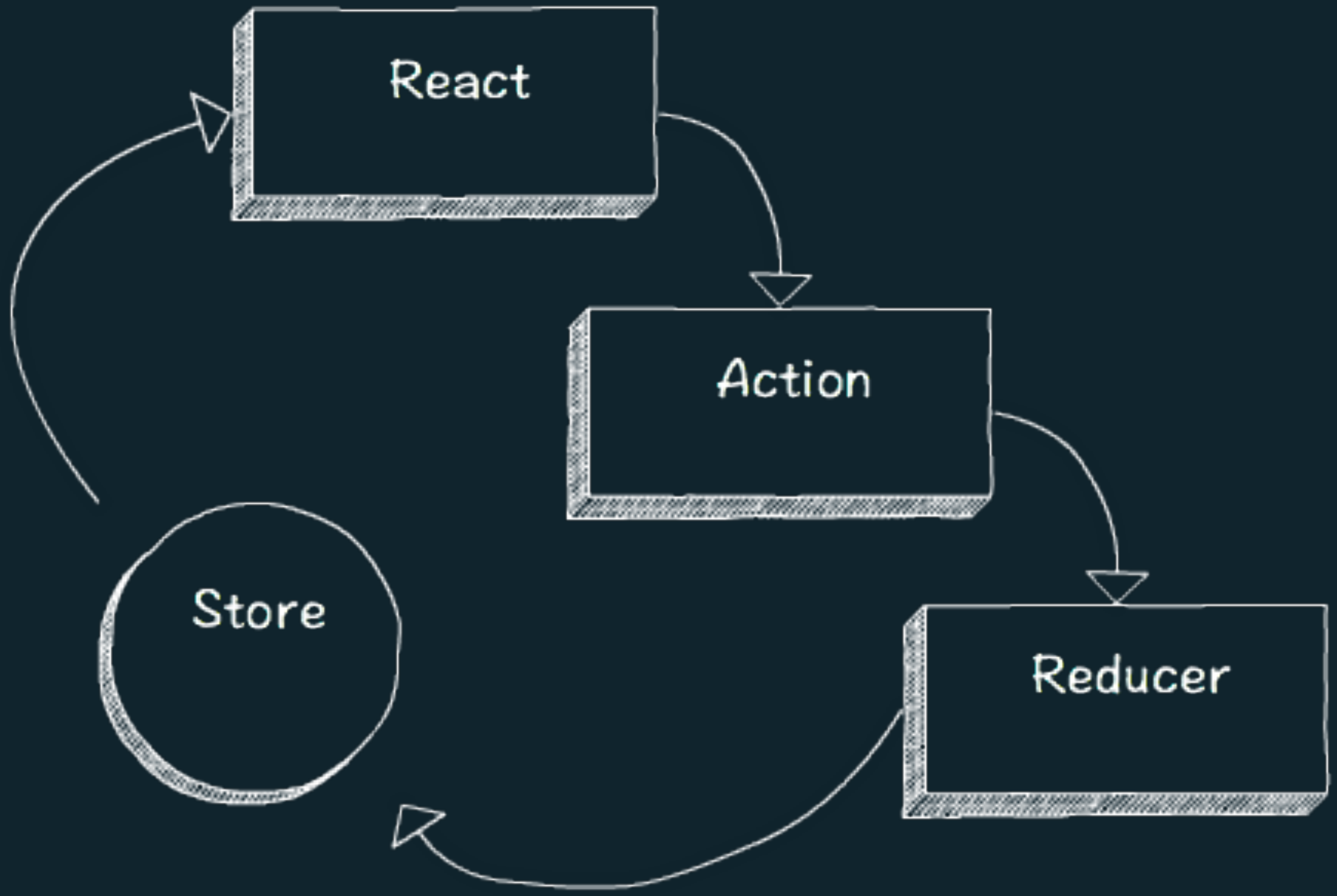
```
function MyForm(props) {
  return(
    // ...
    <Button callToAction="Click here!" />
  )
}
```

`props` **are the arguments passed to a component from the component above it.**

State Management

State Management

Data Flow



State Management

Data Flow

```
import { inputReducer } from 'components/Form'

const store = createStore({
  input: inputReducer,
})
```

State Management

Data Flow

```
export function inputReducer(state, action) {
  switch (action.type) {
    case INPUT_CHANGE:
      return {
        value: action.value,
        ...state,
      }
    default:
      return state;
  }
}
```

State Management

Data Flow

```
// Input.actions.js
const INPUT_CHANGE = 'INPUT_CHANGE';

export function handleChange(event) {
  return {
    type: INPUT_CHANGE,
    value: event.target.value,
  }
}
```

State Management

Data Flow

```
// Input.js
import { handleChange } from './actions';

function Input(props) {
  return (
    <input onChange={(event) => handleOnChange(event)}></input>
  )
}

function mapStateToProps(state) {
  return { input } = state;
}

function mapDispatchToProps(dispatch) {
  return({
    handleOnChange: (event) => {
      dispatch(handleChange(event));
    }
  })
}

export default connect(mapStateToProps, mapDispatchToProps)(Input)
```

State Management

Cohesion

```
src/
|__ components/
|   |__ Form/
|       |__ Input/
|       |__ Form.js
|       |__ Form.actions.js
|       |__ Form.reducers.js
|       |__ index.js
|__ store.js
|__ index.js
```

State Management

Cohesion

Q: How do components talk to each other?
A: Actions!

State Management

Cohesion

```
// Input.js
import { handleChange } from '../components/OtherComponent';

function Input(props) {
  // ...
}

function mapDispatchToProps(dispatch) {
  return({
    hadleOnChange: (event) => {
      dispatch(handleChange(event));
    }
  })
}
```

State Management

Cohesion

Sagas to the rescue!

```
// formAndOtherComponentSagas.js

function* someSagaName() {
  while (true) {
    const payload = yield take('INPUT_CHANGE')
    put({
      type: 'SOME_OTHER_COMPONENT_ACTION',
      data: payload.data,
    })
  }
}
```

State Management

Cohesion

```
src/
|__ sagas/
|__ components/
|   |__ Form/
|       |__ Input/
|       |__ Form.js
|       |__ Form.actions.js
|       |__ Form.reducers.js
|       |__ index.js
|__ store.js
|__ index.js
```

Next Steps

Next Steps

Composition

```
function Table(props) {
  return <table></table>;
}

function MyUniqueTable(props) {
  // do some things with props
  return <Table someProp={specialValue} ...props />;
}
```

Next Steps

Async Actions

```
// Input.sagas.js
function* inputSaga() {
  while (true) {
    const payload = yield take('INPUT_CHANGE');
    const response = yield fetch('/some/api/endpoing', {value: payload.value});
    if (response.ok?) {
      put({type: 'REQUEST_SUCCESS', newValue: response.value});
    } else {
      put({type: 'REQUEST_FAILED', error: response.error});
    }
  }
}
```

Next Steps

GraphQL & Apollo

```
@graphql(
  query ExampleQuery {
  user {
    firstName
    lastName
  }
})
class ExampleContainer extends Component {
  render() {
    const { data: { loading, user } } = this.props;

    if (loading && !user) {
      return <NoDataComponent />;
    }

    return <Example {...user} />;
  }
}

export default ExampleContainer;
```

Try it Yourself

1. Clone the repo (github.com/benortiz/talk-react-crash-course).
2. Create a `<HelloWorld />` component.
3. Use JSON Placeholder to create your own async component.