JavaScript Programming - (9)

- UI 개발방법에 대한 분석
  - jQuery-like Web Development
  - Object-Oriented Web Development
- React
  - Declarative
  - Component-Based
  - Mental Model

먼저, 왜 React를 사용하는지 이해해보자

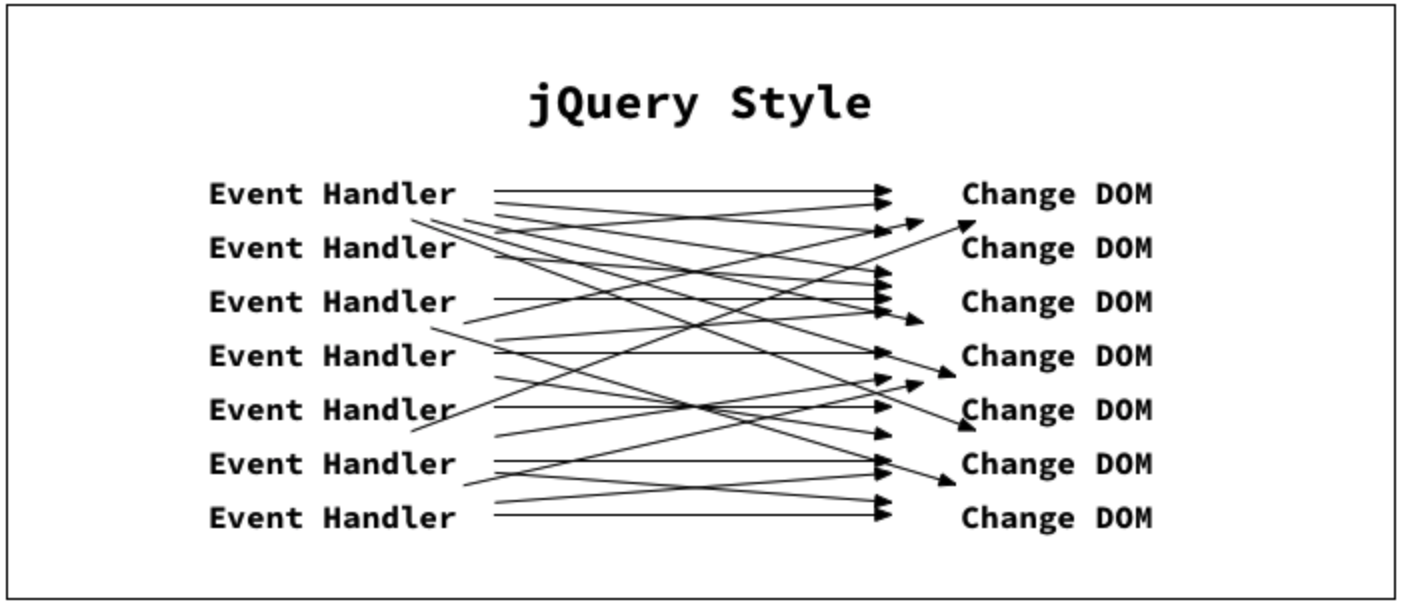
- 언제, 어디서 → 무엇을 → "왜" → 어떻게
- UI 개발이 어떻게 진행됐는지 살펴보고 어떤 문제가 있었는지 알아보자

jQuery-like Web Development

```
function init() {
  const button = $('#button');
  // ...

  button.on('click', buttonClickHandler);
   // ...
}

function buttonClickHandler() {/* ... */}
function someClickHandler() {/* ... */}

$(document).on('DOMContentLoaded', init);
```

jQuery-like Web Development

- Event Handler → Change DOM
- 하지만 사실은 아래처럼 되버린다.



- 데이터가 언제, 어떻게, 어디서 변했는지 알기 힘들다.
- 프로그래밍은 데이터를 어떻게 관리하고 처리할 것인지에 초점을 두는데 반해 이벤트(event) 기반은 이 데이터 관리를 잘하기 어렵게 만든다.

jQuery-like Web Development

- 가장 큰 문제점은 뷰(HTML)가 변하면 거의 모든 코드가 흔들린다.

```
const container = document.getElementById('#container');
const history = container.getElementById('#history');
const currentHistoryId = history.dataset.id;
// ...

const p = document.createElement('p');
p.id = 'status';
container.appendChild(p);
// ...
```

- 뷰가 변경되서 HTML 구조가 변경된다면?
  - `#history`가 `#container` 밖으로 나가게 된다면?
  - `#status`가 `#container` 밖으로 나가게 된다면?

jQuery-like Web Development

- 데이터와 뷰가 엮이기 쉬우니 같거나 비슷한 버튼이라도 중복적으로 작성되는 코드가 많아지게 된다. → 유지보수의 지옥

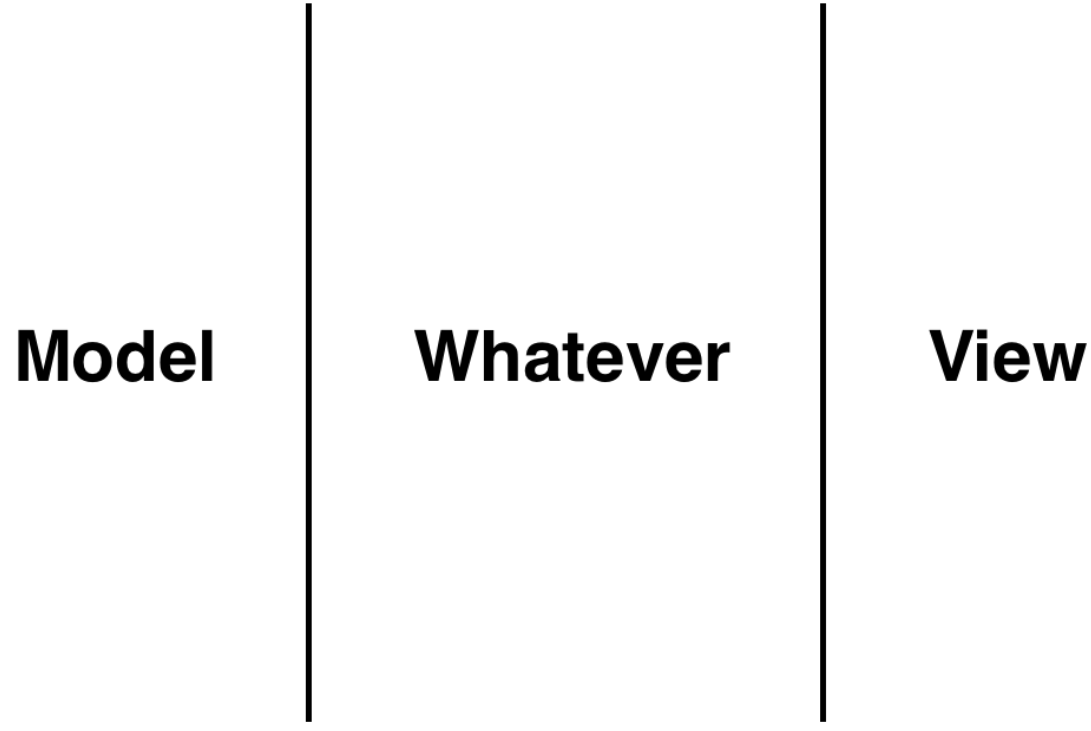
Solution for jQuery-like

- 데이터와 뷰를 최대한 분리하자
- **잘 변하지 않는 것(데이터)** 에서 **잘 변하는 것(뷰)** 을 최대한 분리하자

MVW (`M` odel - `V` iew - `W` hatever)

> 객체지향 프로그래밍을(OOP) 활용

- MVC (Model - View - "Controller")
- MVP (Model - View - "Presenter")
- MVVM (Model - View - "View Model")

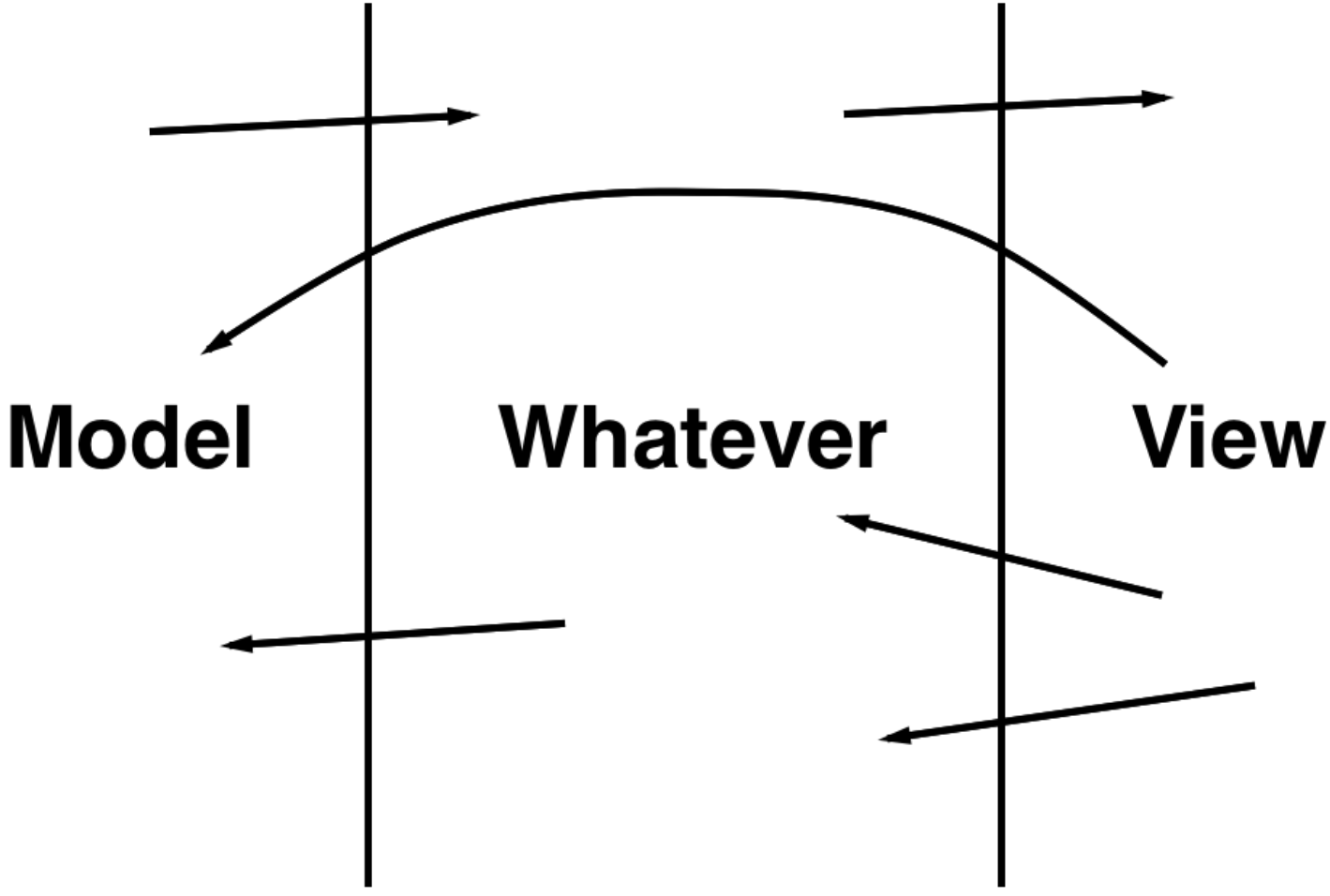


```
class Form extends TraditionalObjectOrientedView {
  // ...

  render() {
    // Read some data passed to the view
    const { isSubmitted, buttonText } = this.attrs;

    if (!isSubmitted && !this.button) {
      // Form is not yet submitted. Create the button!
      this.button = new Button({
        children: buttonText,
        color: 'blue'
      });
      this.el.appendChild(this.button.el);
    }

    if (this.button) {
      // The button is visible. Update its text!
      this.button.attrs.children = buttonText;
      this.button.render();
    }
    // ...
  }
}
```

```
class Form extends TraditionalObjectOrientedView {
  render() {
    // ...

    if (isSubmitted && this.button) {
      // Form was submitted. Destroy the button!
      this.el.removeChild(this.button.el);
      this.button.destroy();
    }

    if (isSubmitted && !this.message) {
      // Form was submitted. Show the success message!
      this.message = new Message({ text: 'Success!' });
      this.el.appendChild(this.message.el);
    }
  }
}
```

문제가 해결된 듯 보이지만 아니었다.

> 경감되었을 뿐이지 문제는 여전히 존재한다.



View에 집중해서 살펴보자

- 웹 UI 개발에 대한 분석

웹 UI 개발에 대한 분석 (1)

- 웹 서비스에서 View는 HTML과 CSS의 조합을 의미한다.
- HTML은 부모, 자식, 조상, 손자 구조인 계층 구조로 구성된다.

```
<div class="container">
  <h1 class="heading1">Heading 1</h1>
  <p>
    Paragraph
    <b>One</b>
  </p>
</div>
```

웹 UI 개발에 대한 분석 (2)

- 상위 뷰에서 하위 뷰에 대한 생성, 삭제를 직접 관리해야한다.

```
class Form extends TraditionalObjectOrientedView {
  render() {
    const { isSubmitted, buttonText } = this.attrs;

    if (!isSubmitted && !this.button) {
      this.button = new Button({
        children: buttonText,
        color: 'blue'
      });
      this.el.appendChild(this.button.el);
    }

    if (isSubmitted && this.button) {
      this.el.removeChild(this.button.el);
      this.button.destroy();
    }
  }
}
```

웹 UI 개발에 대한 분석 (3)

- 뷰는 기본적으로 생성되고 나서 삭제가 되기까지 데이터가 변경될 때 마다 업데이트가 되어야한다.

```
class Form extends TraditionalObjectOrientedView {
  render() {
    const { isSubmitted, buttonText } = this.attrs;

    // ...

    if (this.button) {
      this.button.attrs.children = buttonText;
      this.button.render();
    }
    // ...
  }
}
```

웹 UI 개발에 대한 분석 (4)

- 뷰의 업데이트는 데이터가 "변경되는 시점"을 파악해서 업데이트를 위한 코드를 실행해줘야한다.

```
class Form extends TraditionalObjectOrientedView {
  registerEvent() { /**/ }
  handleSubmit() { /**/ }

  render() {
    const { isSubmitted, buttonText } = this.attrs;

    // ...

    if (this.button) {
      this.button.attrs.children = buttonText;
      this.button.render();
    }
    // ...
  }
}
```

웹 UI 개발에 대한 분석 (5)

- 뷰 관련 모든 코드에서 기본적으로 생성, 삭제, 업데이트에 대한 코드가 필요하다.
- 결과적으로 뷰가 수정될 때 뷰의 메소드를 실행하는 다른 코드들도 변경해줘야하며 이는 변경에 취약하다는 결론에 이른다. 😭

웹 UI 개발에 대한 종합 분석

- 웹 서비스에서 View는 HTML과 CSS의 조합을 의미한다.
- HTML은 부모, 자식, 조상, 손자 구조인 계층 구조로 구성된다.
- 상위 뷰에서 하위 뷰에 대한 생성, 삭제를 직접 관리해야한다.
- 뷰는 기본적으로 생성되고 나서 삭제가 되기까지 데이터가 변경될 때 마다 업데이트가 되어야한다.
- 뷰의 업데이트는 데이터가 "변경되는 시점"을 파악해서 업데이트를 위한 코드를 실행해줘야한다.
- 뷰 관련 모든 코드에서 기본적으로 생성, 삭제, 업데이트에 대한 코드가 필요하다.
- 결과적으로 뷰가 수정될 때 뷰의 메소드를 실행하는 다른 코드들도 변경해줘야하며 이는 변경에 취약하다는 결론에 이른다. 😭

React

- Facebook이 관리하고 있는 View 라이브러리

What is React?

> A JavaScript library for building user interfaces

- Declarative
  - XML-like tags syntax
- Component-Based
  - Your components tell React what you want to render – then React will efficiently update and render just the right components when your data changes.
- Learn Once, Write Anywhere

What is React?

> A JavaScript library for building user interfaces

- Declarative : 사람이 이해하기 쉬운(선언적인) 문법을 사용한다.
  - 태그를 기반으로 한다. (`<div>태그</div>`)
- Component-Based : 컴포넌트 기반의 구조를 기반으로 한다.
  - 개발자 → React
    - 어떤 뷰를 그리고 싶습니다. (컴포넌트 구현을 통해 전달)
  - React → 개발자
    - 데이터가 변할 때 효율적으로 뷰를 업데이트 해줄게요.
- Learn Once, Write Anywhere
  - 웹 뿐만 아니라 모바일, 데스크탑에서도 사용 가능하다.

What is React? - Declarative

- 사람이 이해하기 쉬운 문법을 사용한다. (`JSX` , 태그 기반)

```
const name = 'Taylor';
const div = document.createElement('div');
const p = document.createElement('p');

p.textContent = `Hello ${name}`;
div.appendChild(p);
document.body.appendChild(div);
```

```
function HelloMessage(props) {
  const { name } = props;
  return <div><p>Hello {name}</p></div>;
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('root')
);
```

What is React? - Declarative

> 공식 HTML 태그 요소가 아닌 직접 정의한 뷰라도 문법은 같다.

```
function Button(props) {
  const { children, color, onClick } = props;
  return (
    <button style={{ color }} onClick={onClick}>
      {children}
    </button>
  );
}

// <Button color="blue">Submit</Button>
```

What is React? - Declarative

> 직접 정의한 뷰를 또 다시 직접 정의한 뷰의 자손으로 사용할 수 있다.

```
function Form(props) {
  const { isSubmitted, buttonText } = props;
  return (
    <form>
      <label>
        <input type="text" placeholder="Write name" />
      </label>
      {!isSubmitted &&
        <Button color={'blue'}>{buttonText}</Button>
      }
    </form>
  );
}
```

What is React? - Declarative

1. React Button Component 정의

```
function Button(props) {
  const { children, color, onClick } = props;
  return (
    <button style={{ color }} onClick={onClick}>
      {children}
    </button>
  );
}
```

2. JSX를 통해 사용

```
<Button color="blue">Submit</Button>
```

3. JavaScript를 통해 컴포넌트 생성

```
React.createElement(Button, { color: 'blue' }, 'Submit')
```

What is React? - Component-Based

- Demo : https://codesandbox.io/s/ooz32rjky5

```
ReactDOM.render(
  <Calculator buttonText="Hello Button" />,
  document.getElementById('id')
);
```

```
<div>
  Value: 0
  <Button color="blue">Hello Button</Button>
</div>
```

```
<div>
  Value: 0
  <button style={{ color: 'blue' }}>Hello Button</button>
</div>
```

What is React? - Component-Based

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: 0 };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState(prevState => ({
      value: prevState.value + 1
    }));
  }
  render() {
    return (
      <div>
        Value: {this.state.value}
        <Button color="blue" onClick={this.handleClick}>
          {this.props.buttonText}
        </Button>
      </div>
    );
  }
}
```

What is React? - Component-Based

- 특정 뷰의 Event Listener 관련 코드도 동시에 처리할 수 있다.

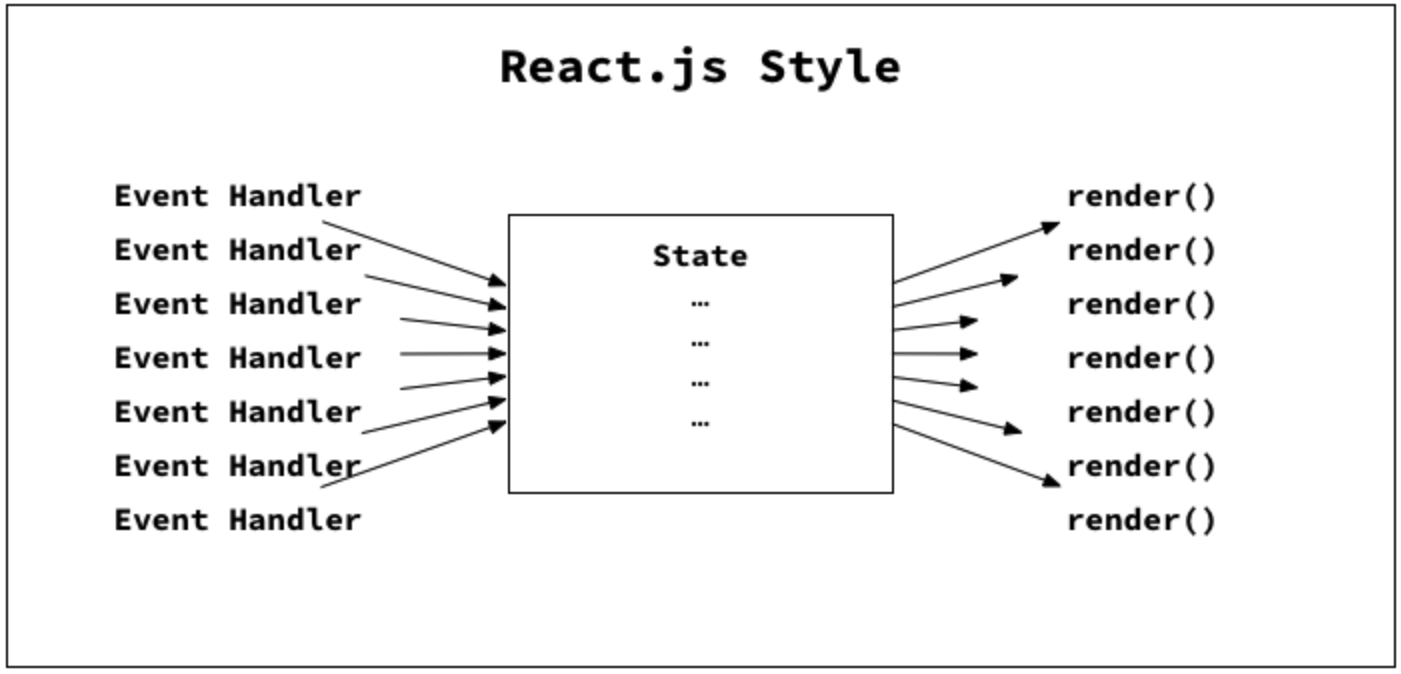
```
class Calculator extends React.Component {
  handeClick() {/* ... */}
  render() {
    return (
      <div>
        Value: {this.state.value}
        <Button color="blue" onClick={this.handleClick}>
          {this.props.buttonText}
        </Button>
      </div>
    );
  }
}
```

What is React? - Component-Based

- 상태 변화가 일어났을 때 관련 하위뷰도 자동으로 업데이트해준다.

```
class Calculator extends React.Component {
  constructor(props) {/* ... */}
  handleClick() {
    this.setState(prevState => ({
      value: prevState.value + 1
    }));
  }
  render() {
    return (
      <div>
        Value: {this.state.value}
        <Button color="blue" onClick={this.handleClick}>
          {this.props.buttonText}
        </Button>
      </div>
    );
  }
}
```

React Web Development



React는 어떤 문제를 해결했을까? (1)

- 웹 서비스에서 View는 HTML과 CSS의 조합을 의미한다.

> HTML, CSS의 코드를 분리 혹은 동시에 작성 가능

- HTML은 부모, 자식, 조상, 손자 구조인 계층 구조로 구성된다.

> JSX를 통해 HTML과 같은 동일한 계층 구조를 유지

- 상위 뷰에서 하위 뷰에 대한 생성, 삭제를 직접 관리해야한다.

> JSX를 통해 React가 알아서 직접 관리

React는 어떤 문제를 해결했을까? (2)

- 뷰는 기본적으로 생성되고 나서 삭제가 되기까지 데이터가 변경될 때 마다 업데이트가 되어야한다.

> React가 자동으로 render 함수를 통해 업데이트해준다.

- 뷰의 업데이트는 데이터가 "변경되는 시점"을 파악해서 업데이트를 위한 코드를 실행해줘야한다.

> React가 `state`, `props`의 변경사항을 감지해서 업데이트해준다.

React는 어떤 문제를 해결했을까? (3)

- 뷰 관련 모든 코드에서 기본적으로 생성, 삭제, 업데이트에 대한 코드가 필요하다.

> 뷰가 어떻게 그려질지에 대한 `render` 함수와 사용자 상호작용에 대한 이벤트 처리 코드만 작성하면 된다.

- 결과적으로 뷰가 수정될 때 뷰의 메소드를 실행하는 다른 코드들도 변경해줘야하며 이는 변경에 취약하다는 결론에 이른다. 😭

> 뷰에서 다른 뷰의 메소드는 호출하지 않으므로 변경이 최소화되며 해당 뷰가 책임지고 있는 계층만 수정해주면 된다.

React를 벗어나서 웹 UI 개발구조를 본다면?

- JavaScript → "HTML + CSS + Event Listener"
- "HTML + CSS + Event Listener" → "JavaScript"

크게 총 2개의 방향으로 나눌 수 있는데 각각은 아래처럼 볼 수 있다.

- Data → View
- View → Data

동일한 데이터를 줄 때 동일한 뷰를 그린다면 버그의 가능성이 훨씬 줄어든다.

React의 의미를 살펴본다면?

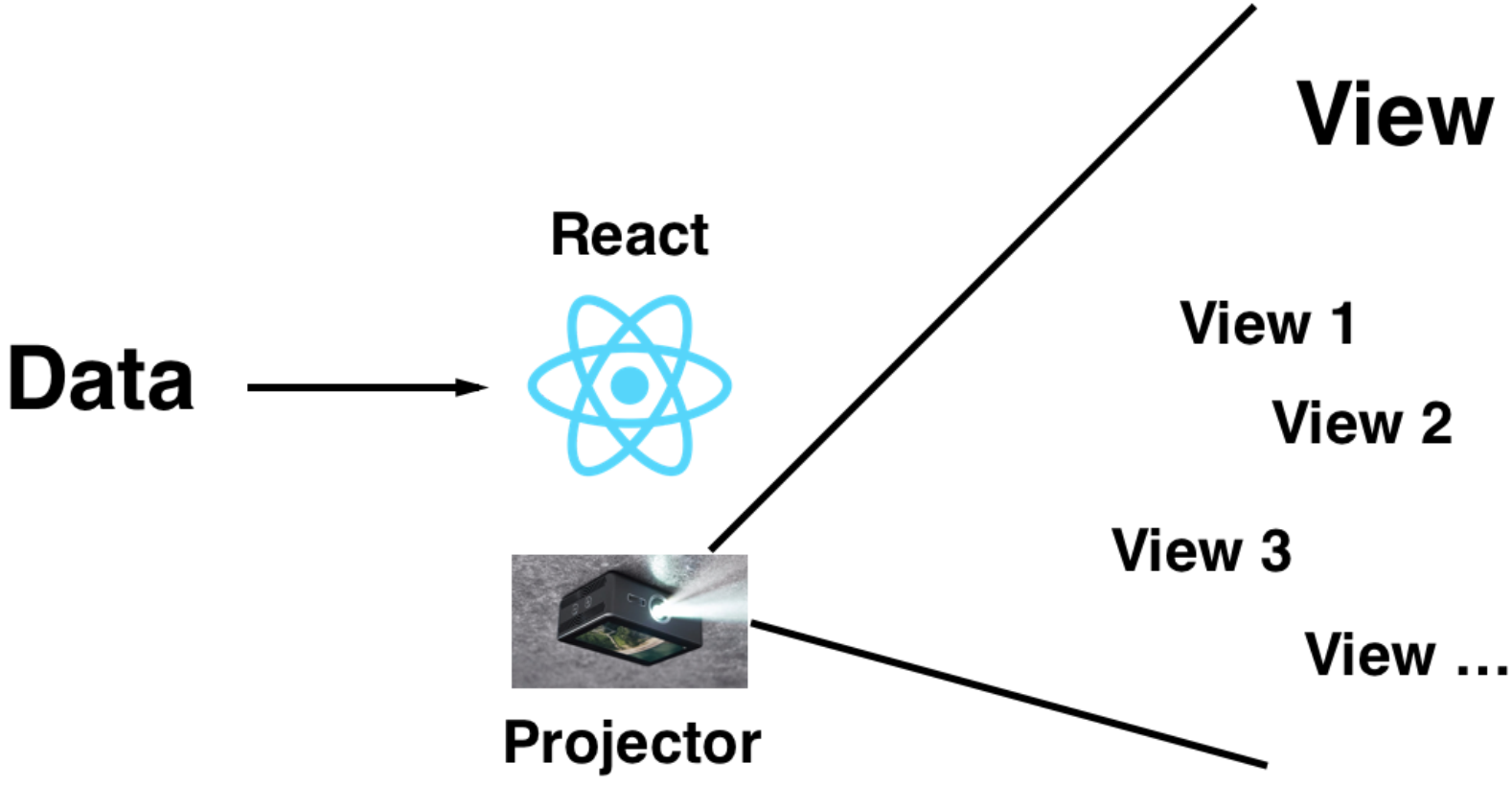
- Data → View
- View → Data

동일한 데이터를 줄 때 동일한 뷰를 그린다면 버그의 가능성이 훨씬 줄어든다.

→ 입력이 동일할 때 출력이 항상 같은 함수와 같은 맥락을 가진다.

React의 의미를 살펴본다면?

- `y = f(x);`
- `View = React(data);`



실습

1. `<Button />` 구현
2. `<HelloWorld />` 구현
3. `<Calculator />` 구현