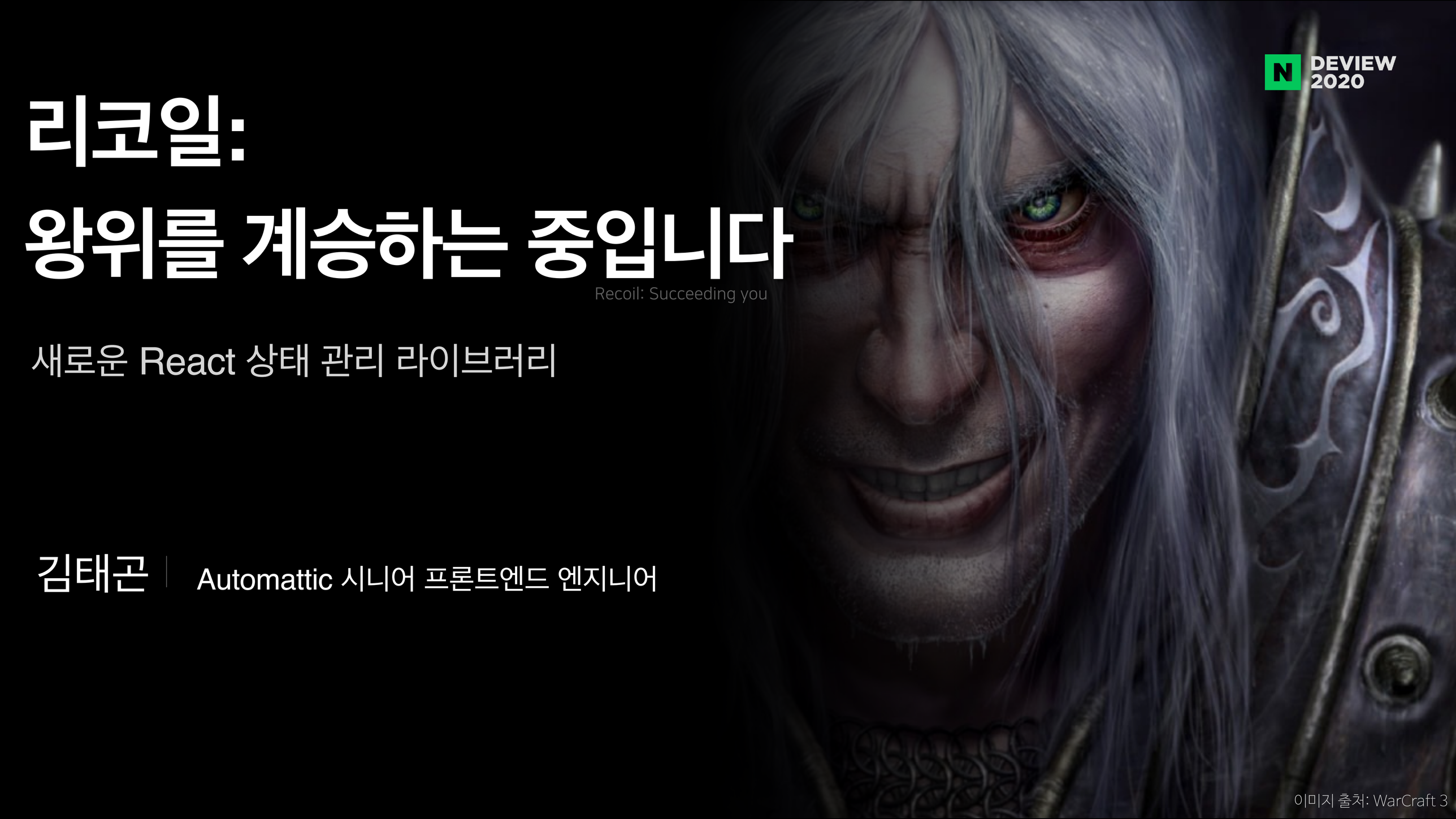
리코일:
왕위를 계승하는 중입니다

Recoil: Succeeding you

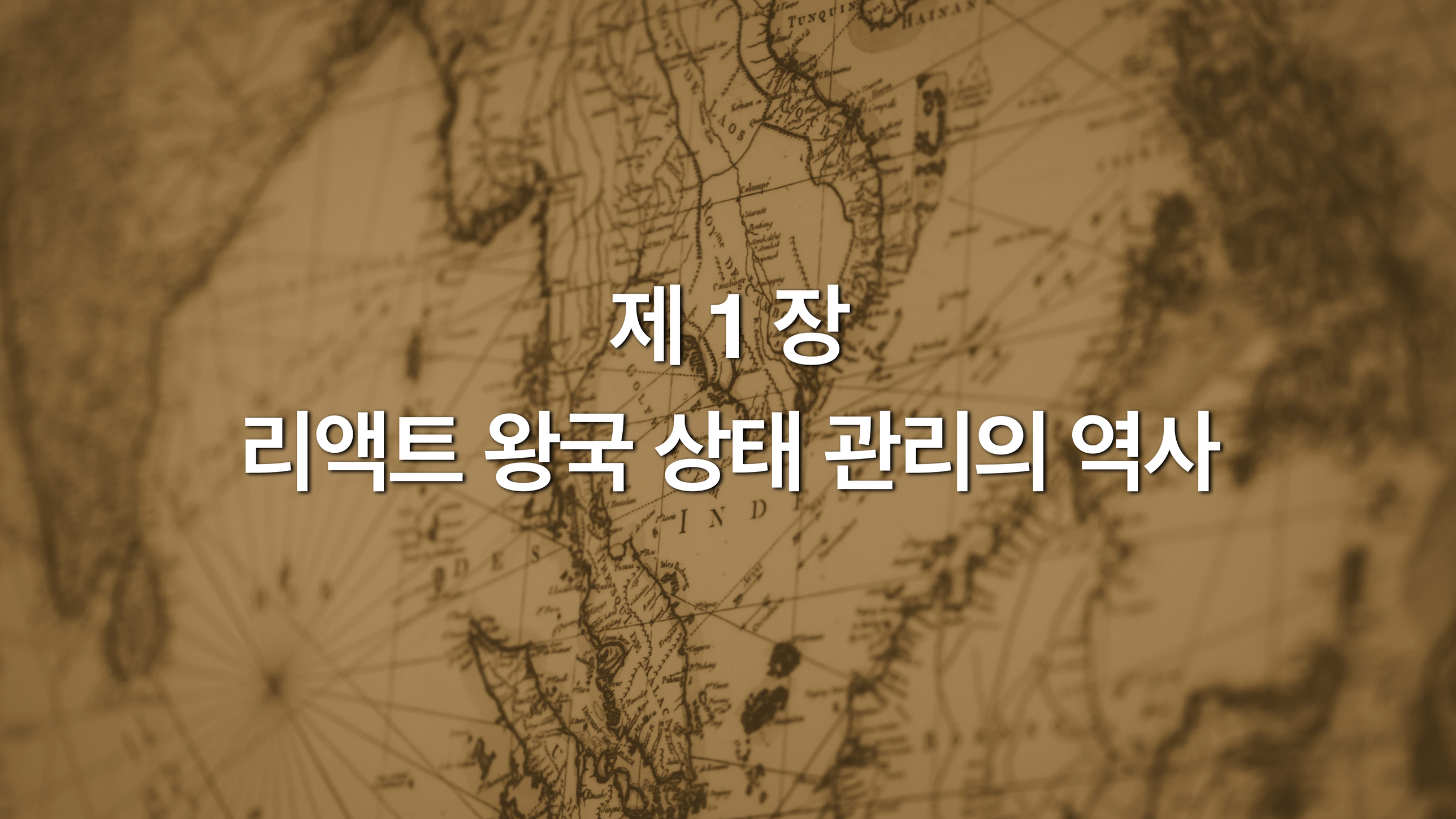
새로운 React 상태 관리 라이브러리

김태곤 | Automattic 시니어 프론트엔드 엔지니어



DEVIEW 2020

이미지 출처: WarCraft 3

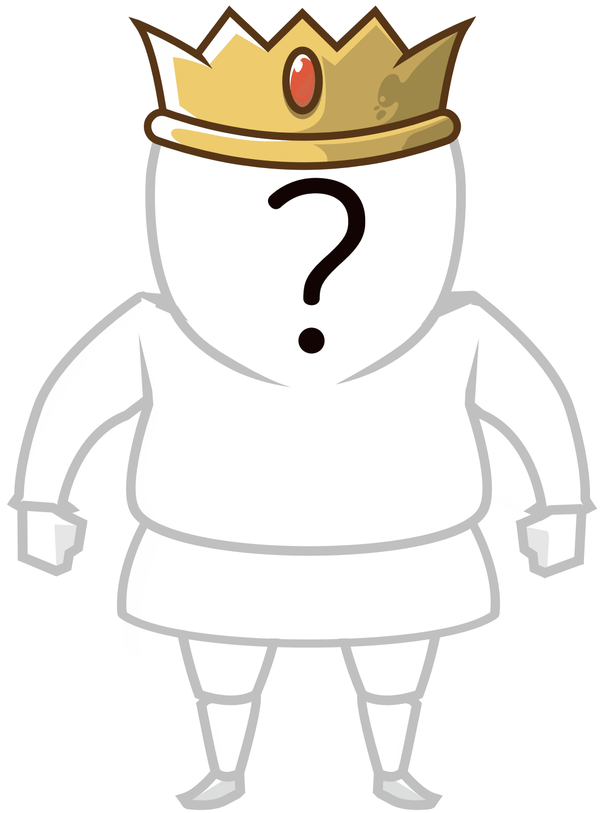


제 1 장
리액트 왕국 상태 관리의 역사

History of React State

DEVIEW 2020

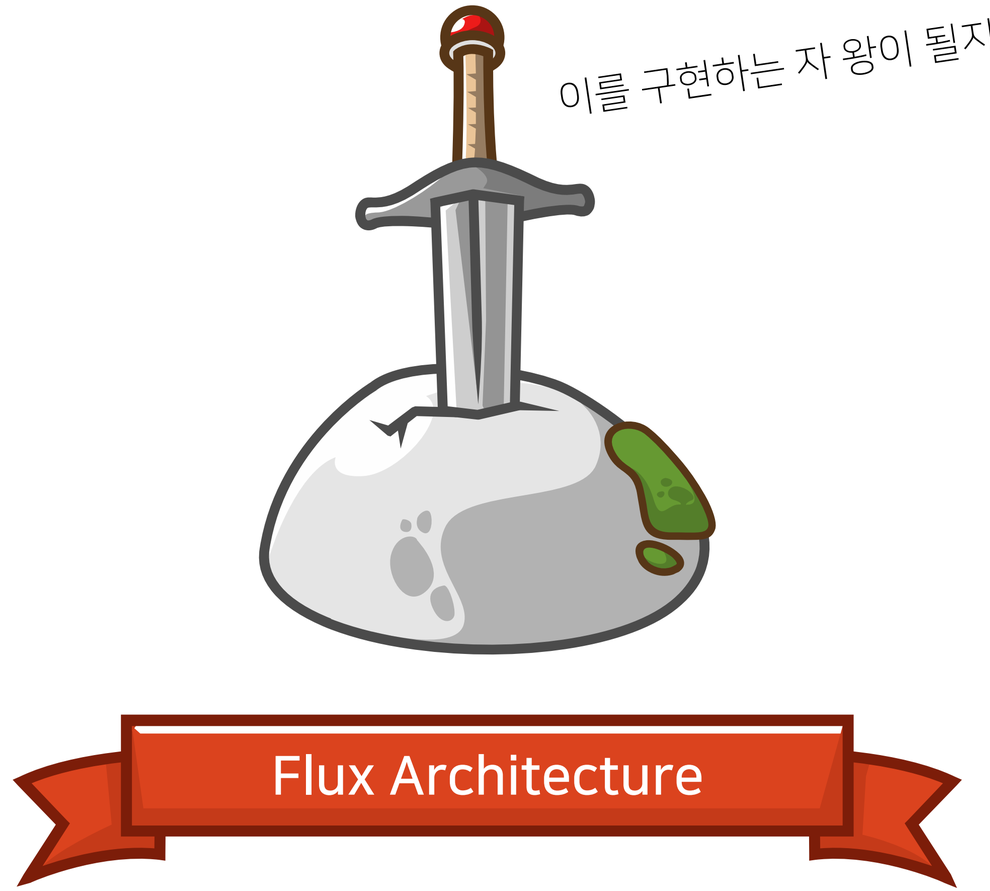
- 왕위 부재:
  한동안 기본적인 context, prop, state가
  유일한 상태 관리 방법



History of React State

DEVIEW 2020

- **왕위 부재:**
  한동안 기본적인 context, prop, state가
  유일한 상태 관리 방법
- **왕의 자격 발표:**
  2014년 Flux architecture

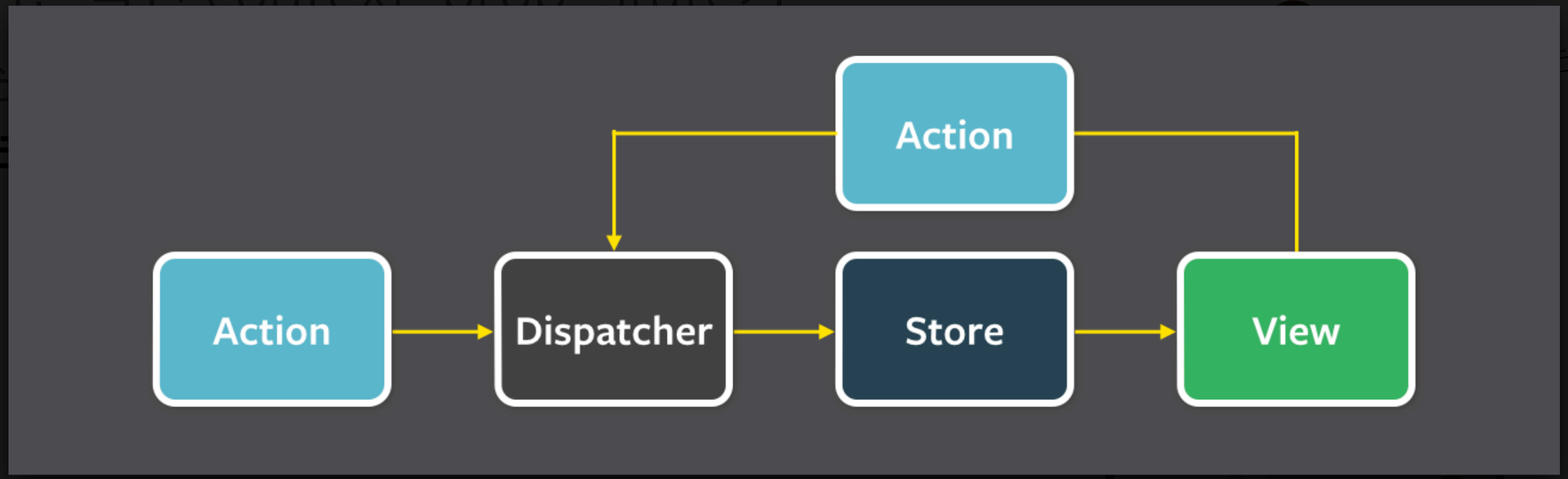


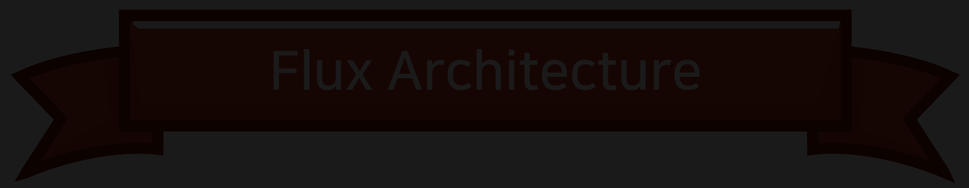
History of React State

DEVIEW 2020

- 왕위 부재:
  한동안 기본적인 context, prop, state가
  유일한 상
- 왕의 자격
  2014년

Flux Architecture

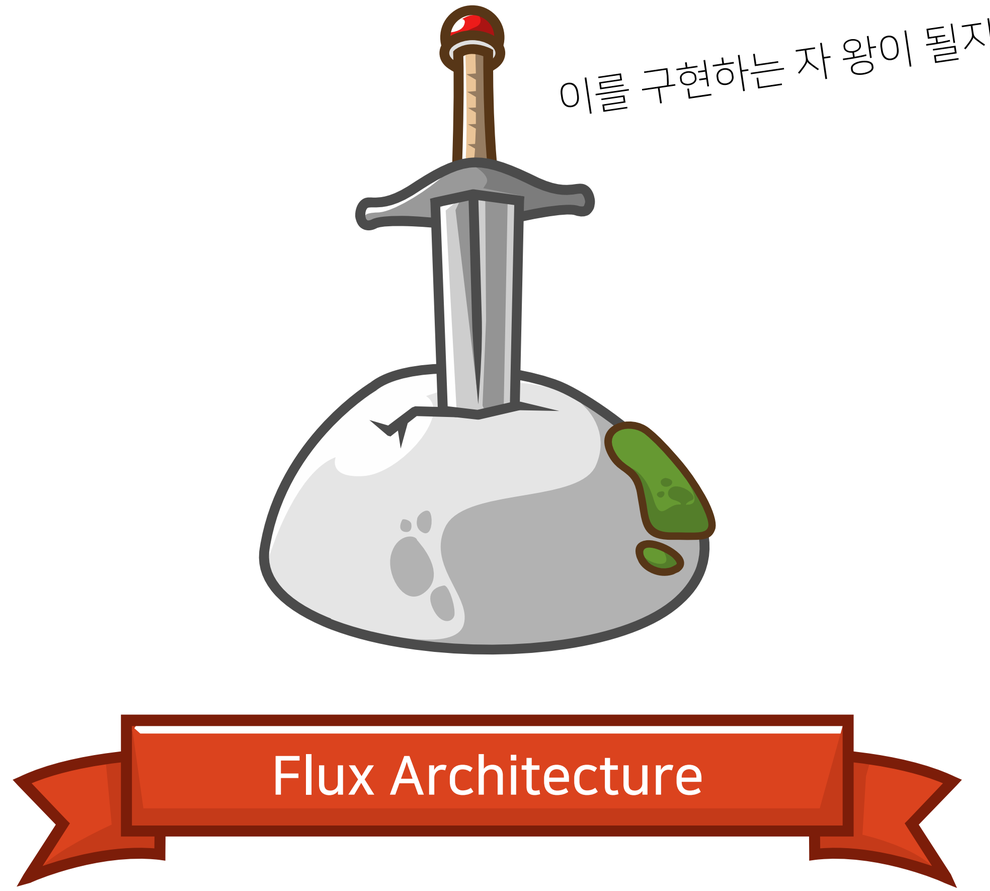




History of React State

DEVIEW 2020

- **왕위 부재:**
  한동안 기본적인 context, prop, state가
  유일한 상태 관리 방법
- **왕의 자격 발표:**
  2014년 Flux architecture



History of React State

- 왕위 부재:
  한동안 기본적인 context, prop, state가 유일한 상태 관리 방법
- 왕의 자격 발표:
  2014년 Flux architecture
- 유력 후보 등장:
  2015년 Redux 릴리스

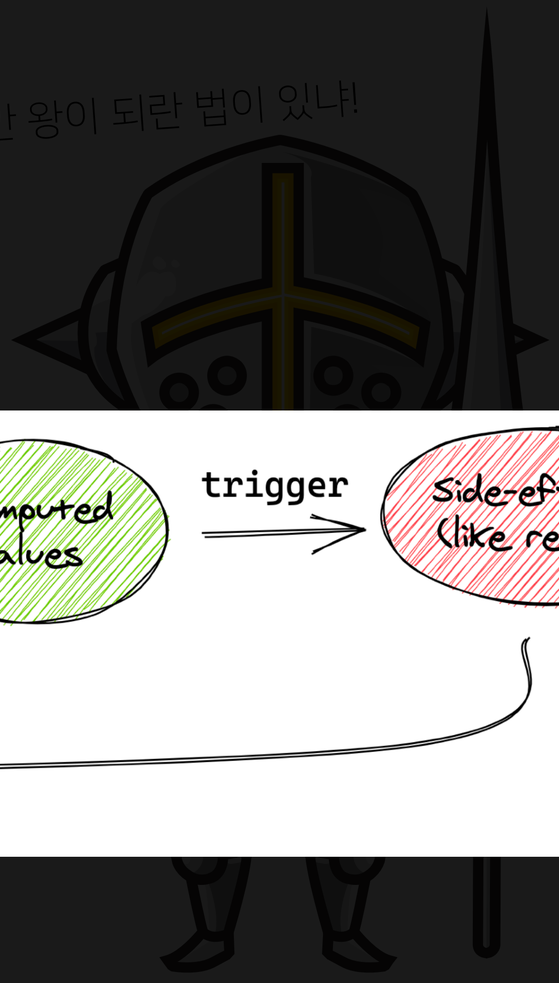
History of React State

- **왕위 부재:**
  한동안 기본적인 context, prop, state가 유일한 상태 관리 방법
- **왕의 자격 발표:**
  2014년 Flux architecture
- **유력 후보 등장:**
  2015년 Redux 릴리스
- **경쟁자 등장:**
  동시기 MobX 릴리스

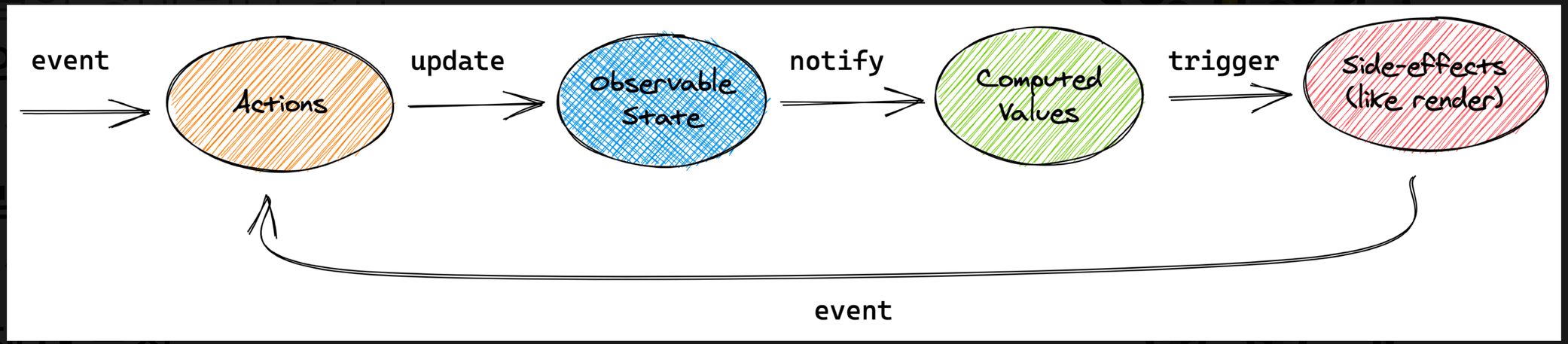
History of React State

DEVIEW 2020

- 왕위 부재:
  한동안 기본적인 context, prop, state가
  유일한 상태 관리 방법
- 왕의
  20
- 유력
  20
- 경쟁자 등장:
  동시기 MobX 릴리스



MobX



History of React State

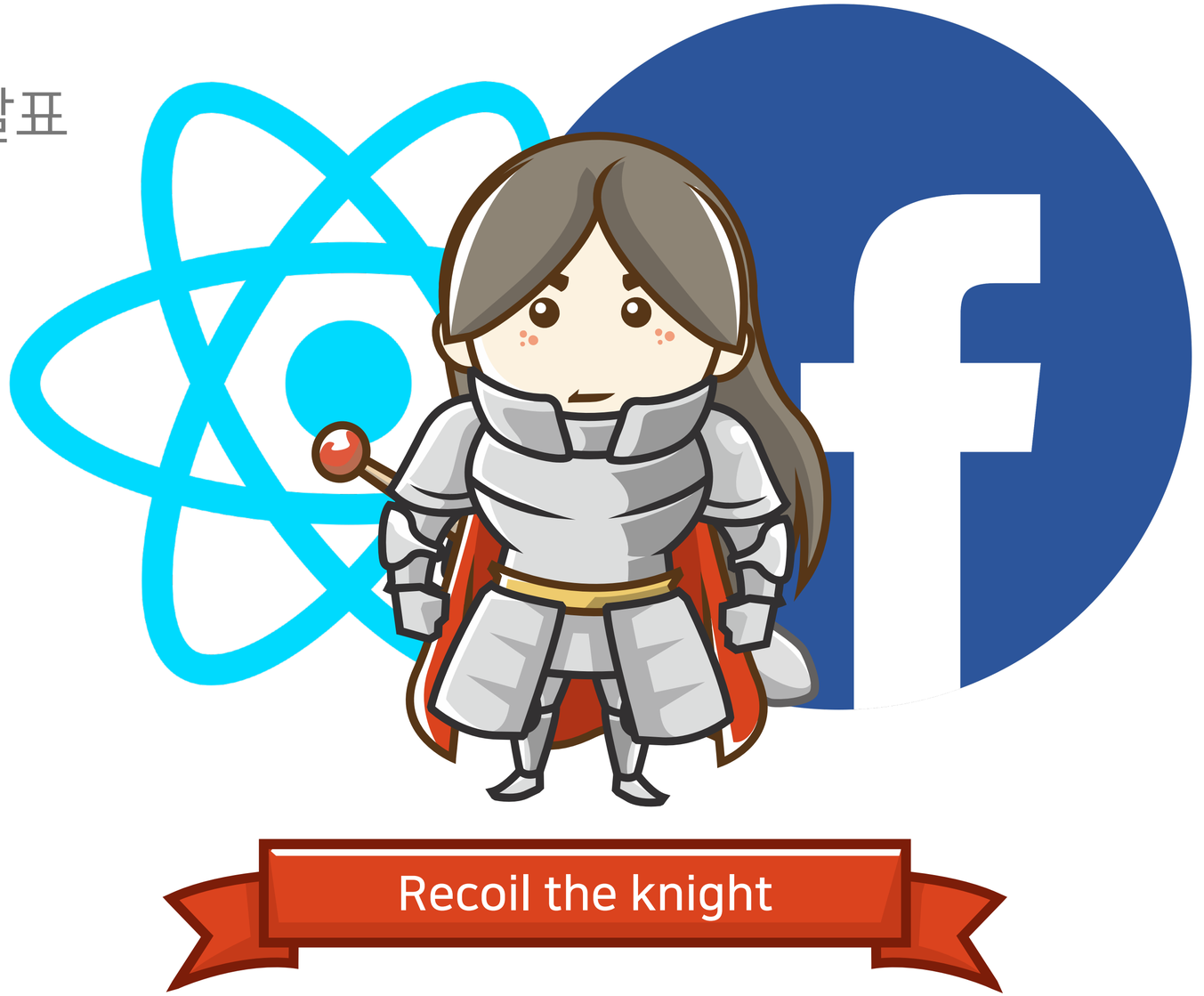
- **왕위 부재:**
  한동안 기본적인 context, prop, state가 유일한 상태 관리 방법
- **왕의 자격 발표:**
  2014년 Flux architecture
- **유력 후보 등장:**
  2015년 Redux 릴리스
- **경쟁자 등장:**
  동시기 MobX 릴리스

제 2 장
리코일의 탄생

Rising of Recoil

DEVIEW 2020

- 페이스북이 2020. 5. React EU에서 발표
- React를 "더 잘" 지원할 목적으로 작성



Context API

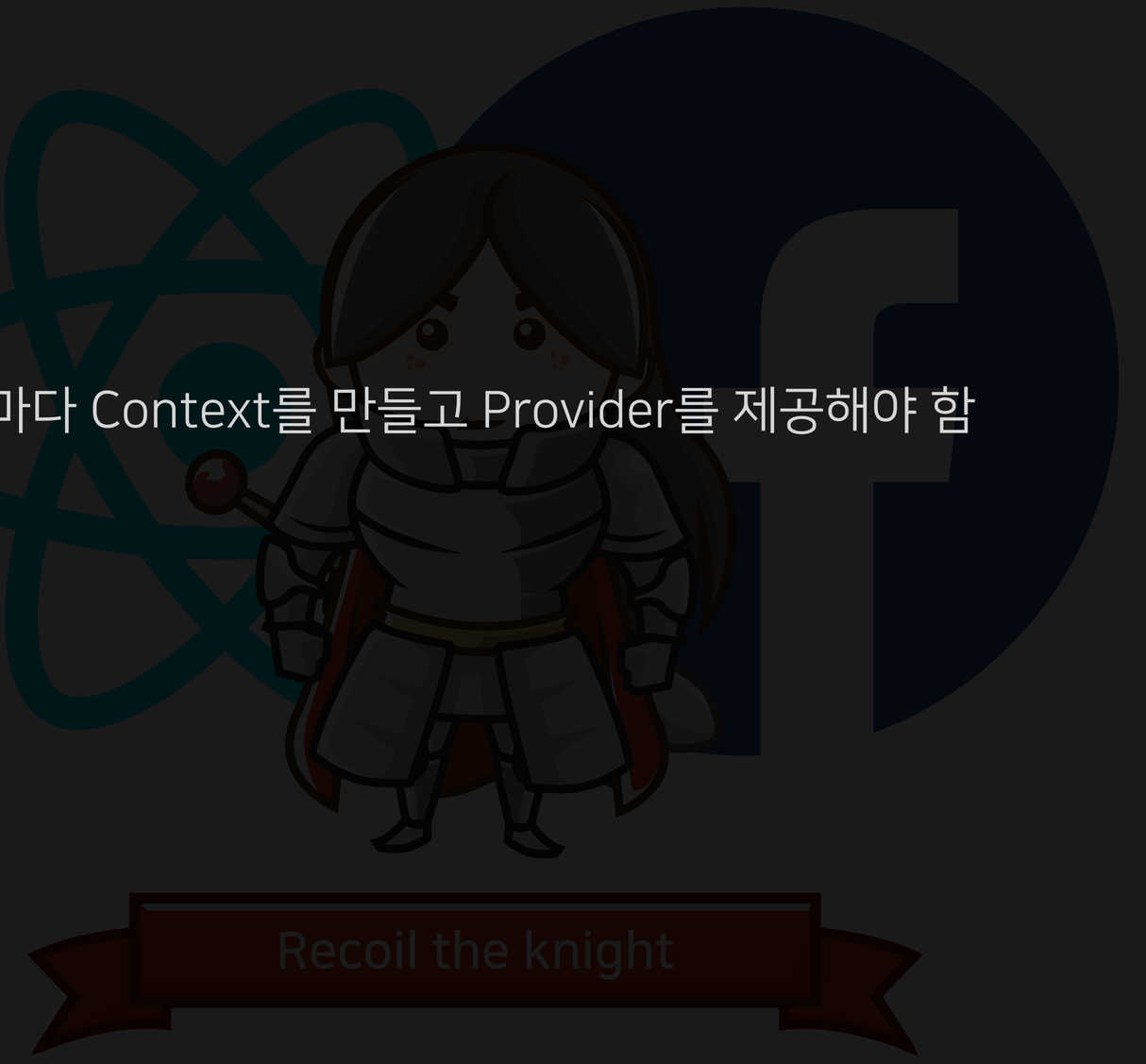
```
// Theme context, default to light theme
const ThemeContext = React.createContext('light');

// Signed-in user context
const UserContext = React.createContext({
  name: 'Guest',
});

class App extends React.Component {
  render() {
    const {signedInUser, theme} = this.props;

    // App component that provides initial context
values
    return (
      <ThemeContext.Provider value={theme}>
        <UserContext.Provider value={signedInUser}>
          <Layout />
        </UserContext.Provider>
      </ThemeContext.Provider>
    );
  }
}
```

데이터마다 Context를 만들고 Provider를 제공해야 함

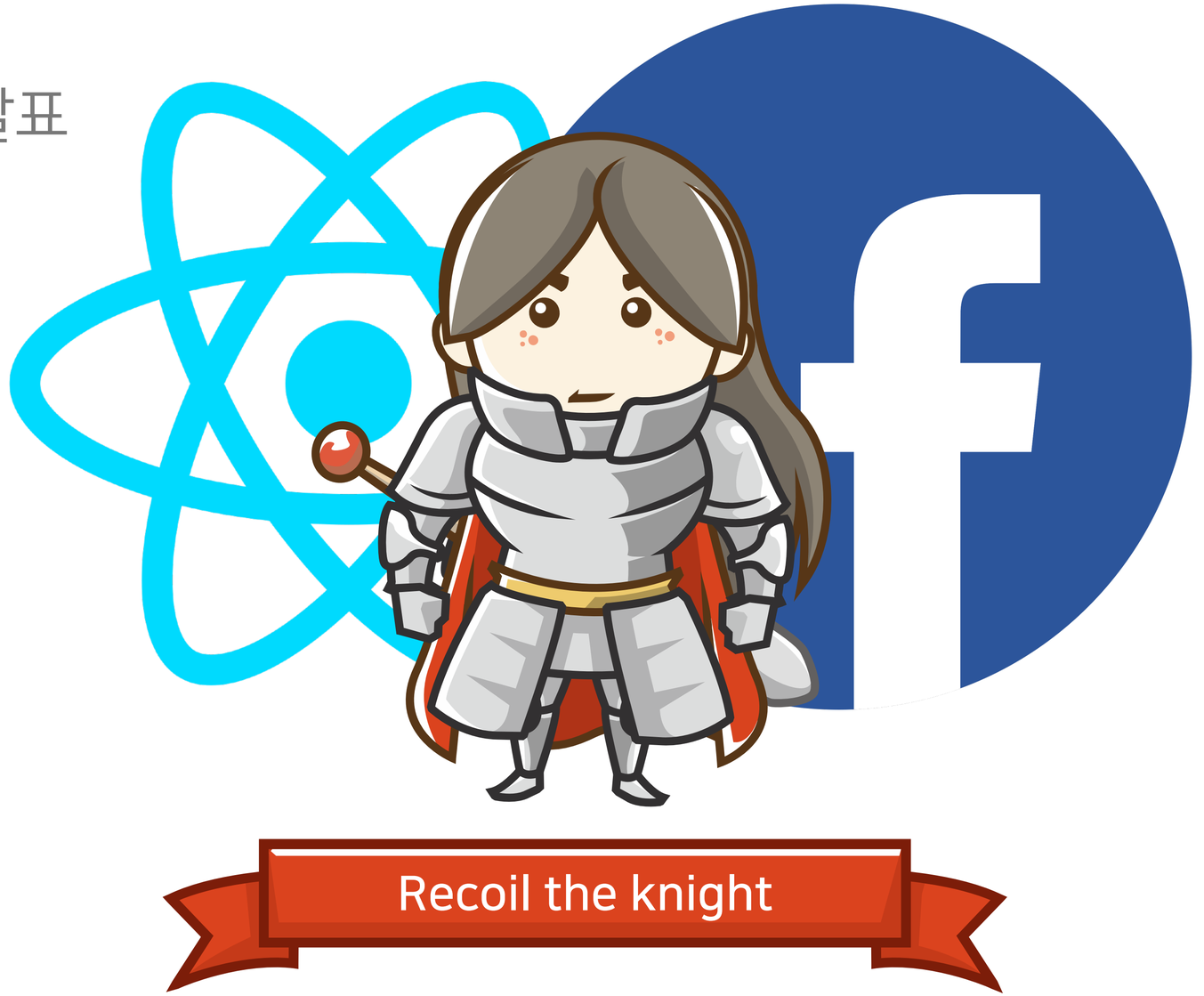



Elements
StripeHookProviderInnerWrapper InjectStripe
StripeHookProviderInnerWrapper
Context.Provider
CompositeCheckout
QuerySitePlans
QuerySitePurchases
ConnectFunction Memo
ReactRedux.Provider
QueryPlans
QueryProductsList
ConnectFunction Memo
ReactRedux.Provider
QueryContactDetailsCache
ConnectFunction Memo
ReactRedux.Provider
PageViewTracker
CartMessages Localized
CartMessages
CartProvider
Context.Provider
CheckoutProvider
CheckoutErrorBoundary
CheckoutProviderPropValidator
ThemeProvider
Context.Consumer
Context.Provider
Context.Provider
LineItemsProvider
Context.Provider
Context.Provider
TransactionStatusHandler
WPCheckout

Rising of Recoil

DEVIEW 2020

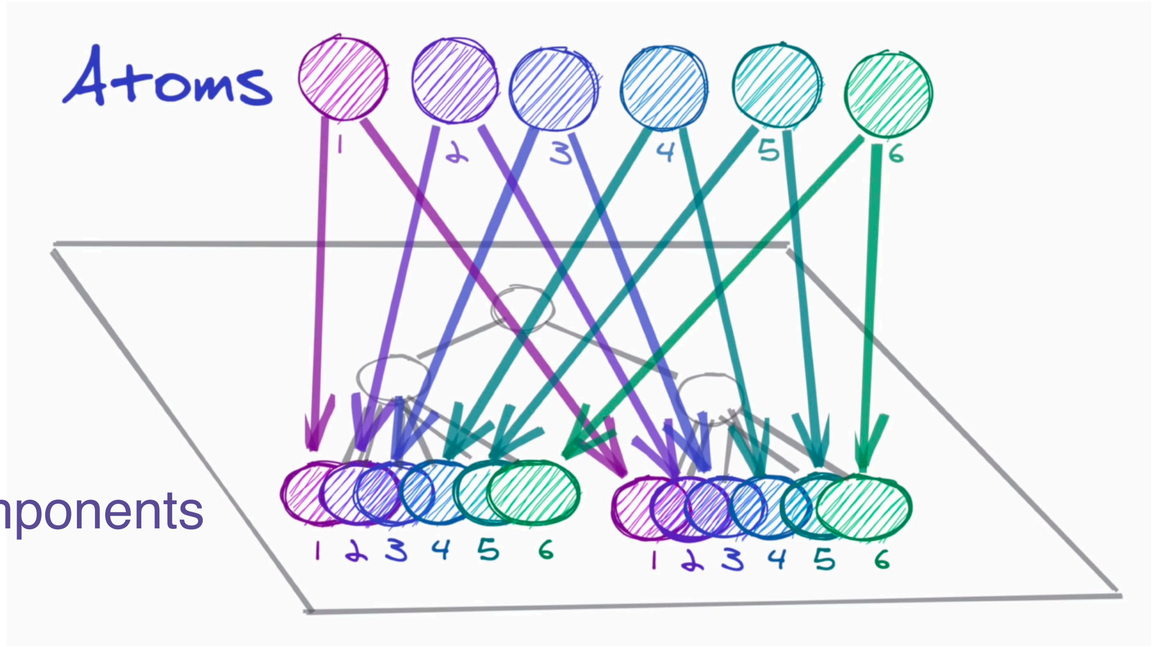
- 페이스북이 2020. 5. React EU에서 발표
- React를 "더 잘" 지원할 목적으로 작성



Rising of Recoil

DEVIEW 2020

- 페이스북이 2020. 5. React EU에서 발표
- React를 "더 잘" 지원할 목적으로 작성
- Data-flow Graph



Rising of Recoil

DEVIEW 2020

- 페이스북이 2020. 5. React EU에서 발표
- React를 "더 잘" 지원할 목적으로 작성
- Data-flow Graph
- React Hooks로만 사용할 수 있음
- 주요 개념
  아톰(atom) : 데이터 조각

```
import { atom, useRecoilState } from 'recoil';

const countState = atom({
  key: 'count',
  default: 0,
});
```

Rising of Recoil

DEVIEW 2020

- 페이스북이 2020. 5. React EU에서 발표
- React를 "더 잘" 지원할 목적으로 작성
- Data-flow Graph
- React Hooks로만 사용할 수 있음
- 주요 개념

  아톰(atom) : 데이터 조각

  셀렉터(selector):

  1. 아톰에서 파생된 데이터 조각
  2. 데이터를 반환하는 순수 함수

```
import { atom, selector } from 'recoil';

const countState = atom({
  key: 'count',
  default: 0,
});

const doubleCount = selector({
  key: 'doubleCount',
  get: ({get}) => {
    const count = get(countState);
    return count * 2;
  }
});

const currentUser = selector({
  key: 'currentUser',
  get: async () => {
    const info = await fetch('server/api/endpoint');
    return info;
  }
});
```

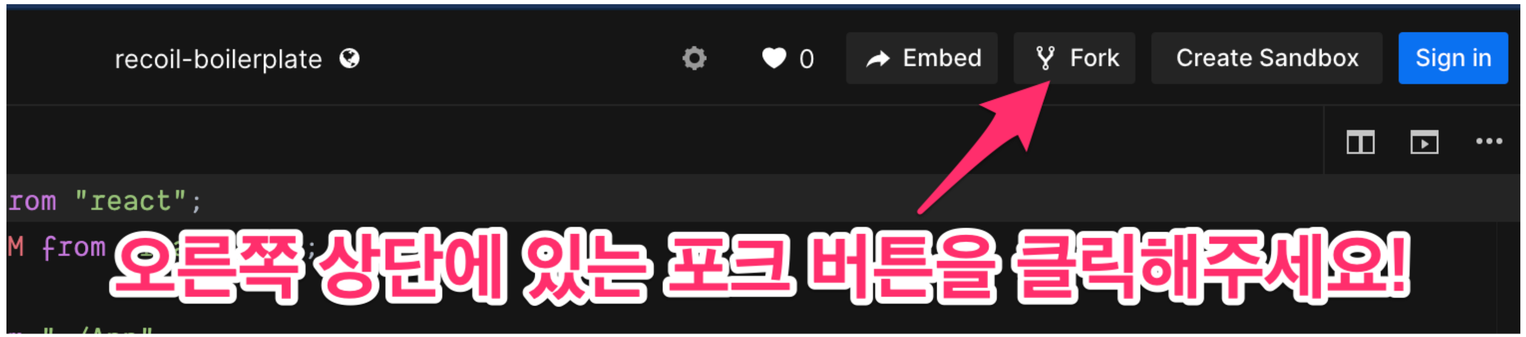
제 3 장
예제로 배워보는 리코일

Learning by Examples

DEVIEW 2020

예제 1. 심플 카운터

아래 보일러 플레이트를 포크해서 사용하세요.
둘 중 아무거나 사용하셔도 됩니다.
https://bit.ly/348ooYD
https://codesandbox.io/s/recoil-boilerplate-ne6h4



1분 후에 시작합니다.

Learning by Examples

DEVIEW 2020

예제 1. 심플 카운터

- App.jsx 파일에 <RecoilRoot>를 추가합니다.

App.jsx

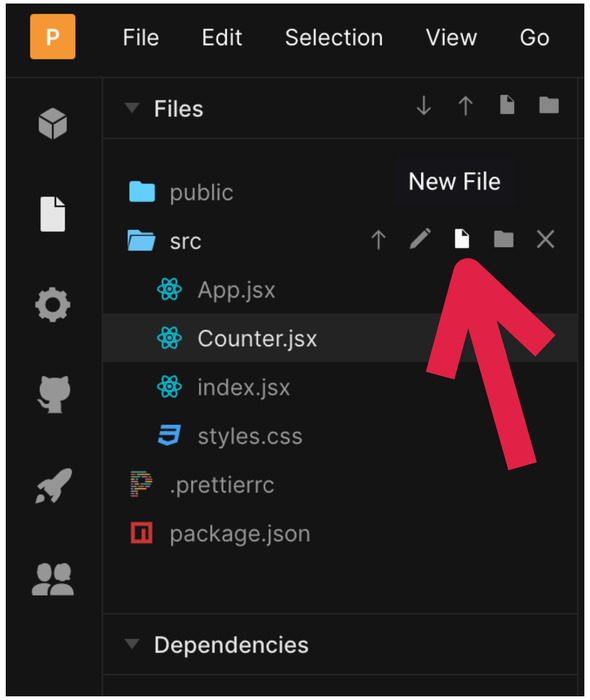
```
1   import React from "react";
2   import { RecoilRoot } from "recoil";
3   import "./styles.css";
4
5   export default function App() {
6     return (
7       <div className="App">
8         <h1>Recoil 카운터</h1>
9         <p>아래 버튼을 클릭해보세요.</p>
10        <RecoilRoot>
11        </RecoilRoot>
12      </div>
13    );
14  }
15
```

Learning by Examples

DEVIEW 2020

예제 1. 심플 카운터

- Counter.jsx 파일을 만들고 빈 Counter 컴포넌트를 작성합니다.
- App.jsx 파일에 Counter 컴포넌트를 추가합니다.



App.jsx Counter.jsx

```jsx
import React from "react";

export default function Counter() {
  return (
    <div className="counter">
    </div>
  );
}

```

App.jsx Counter.jsx

```jsx
import React from "react";
import { RecoilRoot } from "recoil";
import Counter from "./Counter";
import "./styles.css";

export default function App() {
  return (
    <div className="App">
      <h1>Recoil 카운터</h1>
      <p>아래 버튼을 클릭해보세요.</p>
      <RecoilRoot>
        <Counter />
      </RecoilRoot>
    </div>
  );
}

```

Learning by Examples

DEVIEW 2020

예제 1. 심플 카운터

- Counter.jsx에 count 상태 아톰을 선언합니다.

Counter.jsx

```
1  import React from 'react';
2  import { atom } from 'recoil';
3
4  const countState = atom({
5    key: 'countState',
6    default: 0
7  });
8
9  export default function Counter() {
10   return (
11     <div className="counter">
12     </div>
13   );
14 }
15
```

Learning by Examples

DEVIEW 2020

예제 1. 심플 카운터

- count 아톰을 사용하기 위해 useRecoilState 훅을 호출합니다.

Counter.jsx

```jsx
import React from 'react';
import { atom, useRecoilState } from 'recoil';

const countState = atom({
  key: 'countState',
  default: 0
});

export default function Counter() {
  const [count, setCount] = useRecoilState(countState);

  return (
    <div className="counter">
    </div>
  );
}
```

Learning by Examples

DEVIEW 2020

예제 1. 심플 카운터

- 화면에 카운터를 표시하고 카운터를 증가/감소하는 버튼을 추가한다.

Counter.jsx

```jsx
import React from 'react';
import { atom, useRecoilState } from 'recoil';

const countState = atom({
  key: 'countState',
  default: 0
});

export default function Counter() {
  const [count, setCount] = useRecoilState(countState);

  return (
    <div className="counter">
      Count: {count}
      <br />
      <button onClick={() => setCount(count - 1)}>1 감소</button>
      <button onClick={() => setCount(count + 1)}>1 증가</button>
    </div>
  );
}
```

Learning by Examples

예제 1. 심플 카운터

- Recoil 버전: https://codesandbox.io/s/recoil-counter-8y3cn
- Redux 버전: https://codesandbox.io/s/redux-counter-ie750

Learning by Examples

DEVIEW 2020

예제 1. 심플 카운터

- Redux와 비교하면 코드의 양이 확연히 다른 것을 알 수 있다.

with Recoil

Counter.jsx

```jsx
import React from 'react';
import { atom, useRecoilState } from 'recoil';

const countState = atom({
  key: 'countState',
  default: 0
});

export default function Counter() {
  const [count, setCount] = useRecoilState(countState);

  return (
    <div className="counter">
      Count: {count}
      <br />
      <button onClick={() => setCount(count - 1)}>1 감소</button>
      <button onClick={() => setCount(count + 1)}>1 증가</button>
    </div>
  );
}
```

with Redux

App.jsx Counter.jsx

```jsx
import React from 'react';
import { useDispatch, useSelector } from 'react-redux';

// Action type names
const UPDATE = 'UPDATE';

// Action creators
function update(payload) {
  return { type: UPDATE, payload };
}

// Reducer
export function reducer(state = { count: 0 }, action) {
  switch (action.type) {
    case UPDATE:
      return {
        ...state,
        count: state.count + action.payload
      };
    default:
      return state;
  }
}

export default function Counter() {
  const count = useSelector((state) => state.count);
  const dispatch = useDispatch();

  return (
    <div className="counter">
      Count: {count}
      <br />
      <button onClick={() => dispatch(update(-1))}>1 감소</button>
      <button onClick={() => dispatch(update(1))}>1 증가</button>
    </div>
  );
}
```

Learning by Examples

DEVIEW 2020

예제 1. 심플 카운터

- 코드를 보면 Recoil이 조금 더 직관적이다.

with Recoil

```
 9  export default function Counter() {
10    const [count, setCount] = useRecoilState(countState);
11
12    return (
13      <div className="counter">
14        Count: {count}
15        <br />
16        <button onClick={() => setCount(count - 1)}>1 감소</button>
17        <button onClick={() => setCount(count + 1)}>1 증가</button>
18      </div>
19    );
20  }
```

with Redux

```
25  export default function Counter() {
26    const count = useSelector((state) => state.count);
27    const dispatch = useDispatch();
28
29    return (
30      <div className="counter">
31        Count: {count}
32        <br />
33        <button onClick={() => dispatch(update(-1))}>1 감소</button>
34        <button onClick={() => dispatch(update(1))}>1 증가</button>
35      </div>
36    );
37  }
```

Learning by Examples

DEVIEW 2020

예제 2. 심플 카운터 + 홀짝 셀렉터

- 아까 만든 예제를 조금 수정할 겁니다.
- 셀렉터(selector)에 대해 복습
  1. 아톰에서 파생된 데이터 조각
  2. 데이터를 반환하는 순수 함수

Learning by Examples

DEVIEW 2020

예제 2. 심플 카운터 + 홀짝 셀렉터

- oddEven 셀렉터를 countState 아톰 아래에 작성해줍니다.

Counter.jsx

```
1  import React from 're';
2  import { atom, selector, useRecoilState } from 'recoil';
3
4  const countState = atom({
5    key: 'countState',
6    default: 0
7  });
8
9  const oddEvenState = selector({
10   key: 'oddEvenState',
11   get: ({get}) => {
12     const count = get(countState);
13     return count % 2 ? '홀' : '짝';
14   }
15 });
```

Learning by Examples

DEVIEW 2020

예제 2. 심플 카운터 + 홀짝 셀렉터

- Read-only 데이터에 사용하는 Hook이 따로 있다.
- atom과 selector는 구분없이 동일한 Hooks를 사용해서 다룬다.

```
1  import React from 'react';
2  import { atom, selector, useRecoilState, useRecoilValue } from 'recoil';
```

```
17 export default function Counter() {
18   const [count, setCount] = useRecoilState(countState);
19   const oddEven = useRecoilValue(oddEvenState);
20
21   return (
22     <div className="counter">
23       Count: {count} / 홀짝: {oddEven}
24       <br />
25       <button onClick={() => setCount(count - 1)}>1 감소</button>
26       <button onClick={() => setCount(count + 1)}>1 증가</button>
27     </div>
28   );
29 }
```

Learning by Examples

DEVIEW 2020

예제 3. 서버 API 연동

- 서버에서 랜덤한 고양이 사진 주소를 가져와서 표시하는 애플리케이션
- fetch()와 React.Suspense 기능을 사용합니다.
- 타이핑을 줄이기 위해 아래 샌드박스를 포크하세요.
  https://codesandbox.io/s/random-cat-dy2ul

Learning by Examples

DEVIEW 2020

예제 3. 서버 API 연동

- <RecoilRoot>를 <App>에 추가했습니다.
- state.js 파일을 만들고 randomCat이라는 selector를 만들었습니다.

JS state.js

```
1  import { selector } from 'recoil';
2
3  export const randomCat = selector({
4    key: 'randomCat',
5    get: async () => {
6      const response = await fetch('https://aws.random.cat/meow');
7      const data = await response.json();
8
9      return data.file;
10   }
11 });
```

Learning by Examples

DEVIEW 2020

예제 3. 서버 API 연동

- <RecoilRoot>를 <App>에 추가했습니다.
- state.js 파일을 만들고 randomCat이라는 selector를 만들었습니다.
- randomCat은 외부 서버에서 데이터를 가져온 후 반환하는
  async 함수를 getter로 사용합니다.
- RandomCat 컴포넌트는 셀렉터를 통해 이미지 주소를 불러와서 표시합니다.
- 지금까지는 에러가 나는 게 정상입니다.

Learning by Examples

DEVIEW 2020

예제 3. <

- <Reco
- state.j
- rando
  async
- Rando 합니다.
- 지금까

Error

RandomCat suspended while rendering, but no fallback UI was specified.

Add a <Suspense fallback=...> component higher in the tree to provide a loading indicator or placeholder to display.
    in RandomCat (at App.jsx:12)
    in RecoilRoot (at App.jsx:11)
    in div (at App.jsx:8)
    in App (at src/index.jsx:9)
    in StrictMode (at src/index.jsx:8)

▶ 14 stack frames were collapsed.

evaluate
/src/index.jsx:7:9

```
  4 | import App from "./App";
  5 |
  6 | const rootElement = document.getElementById("root");
```

Learning by Examples

DEVIEW 2020

예제 3. 서버 API 연동

- RandomCat 컴포넌트를 React.Suspense로 감싸줍니다.
- fallback prop에는 기다리는 동안 보여줄 컴포넌트를 전달합니다.

```
6   export default function App() {
7     return (
8       <div className="App">
9         <h1>Random Cat</h1>
10        <p>페이지를 새로 고침할 때마다 랜덤한 고양이 사진을 보여줍니다!</p>
11        <RecoilRoot>
12          <React.Suspense fallback={null}>
13            <RandomCat />
14          </React.Suspense>
15        </RecoilRoot>
16      </div>
17    );
18  }
```

Learning by Examples

DEVIEW 2020

예제 3. 서버 A

- RandomCat 컴
- fallback prop

```
6   export default
7     return (
8       <div class
9         <h1>Rand
10        <p>페이지
11        <RecoilR
12          <React
13            <Ran
14          </Reac
15        </Recoil
16      </div>
17    );
18  }
```

Random Cat

페이지를 새로 고침할 때마다 랜덤한 고양이 사진을 보여줍니다!

Learning by Examples

DEVIEW 2020

예제 3. 서버 API 연동

- React.Suspense 없이 사용하는 방법을 배워봅니다.
- 일단 12라인, 14라인의 React.Suspense 부분을 지워주세요.

```
6   export default function App() {
7     return (
8       <div className="App">
9         <h1>Random Cat</h1>
10        <p>페이지를 새로 고침할 때마다 랜덤한 고양이 사진을 보여줍니다!</p>
11        <RecoilRoot>
12
13          <RandomCat />
14
15        </RecoilRoot>
16      </div>
17    );
18  }
```

Learning by Examples

DEVIEW 2020

예제 3. 서버 API 연동

- useRecoilValue을 useRecoilValueLoadable로 바꿉니다.
- use...Loadable 혹은 Loadable 객체를 반환합니다.
- Loadable 객체는...
  .state 프로퍼티를 통해 상태를 확인하고
  .contents 프로퍼티를 통해 실제 콘텐츠를 가져올 수 있습니다.
  (여기서는 사진 URL)
- Loadable.state는 "hasValue", "loading", "hasError" 셋 중 하나 문자열 값

Learning by Examples

DEVIEW 2020

예제 3. 서버 API 연동

```
5   export default function RandomCat() {
6     const photoUrlLoadable = useRecoilValueLoadable(randomCat);
7     let content = null;
8
9     switch (photoUrlLoadable.state) {
10      case 'hasValue':
11        content = <img src={photoUrlLoadable.contents} alt="random cat" />;
12        break;
13      case 'hasError':
14        content = '데이터를 불러오는 중 에러가 발생했습니다.';
15        break;
16      case 'loading':
17      default:
18        content = '...';
19    }
20
21    return <div className="random-cat">{content}</div>;
22  }
```

Learning by Examples

DEVIEW 2020

예제 3. 서버 API 연동 - MobX 예제

```
import { action, makeAutoObservable } from "mobx"

class Store {
    githubProjects = []
    state = "pending" // "pending", "done" or "error"

    constructor() {
        makeAutoObservable(this)
    }

    fetchProjects() {
        this.githubProjects = []
        this.state = "pending"
        fetchGithubProjectsSomehow().then(
            action("fetchSuccess", projects => {
                const filteredProjects = somePreprocessing(projects)
                this.githubProjects = filteredProjects
                this.state = "done"
            }),
            action("fetchError", error => {
                this.state = "error"
            })
        )
    }
}
```

Learning by Examples

DEVIEW 2020

예제 4. 살짝 더 복잡한 서버 API 연동

- 검색어를 입력하면 해당 애니메이션의 목록을 표시하는 애플리케이션
- 검색어에 따른 결과를 캐시합니다.
- 컴포넌트와 스테이트의 폴더를 분리했습니다.
- 아래 샌드박스를 포크하세요.
  https://codesandbox.io/s/anime-search-m9zcc

Learning by Examples

DEVIEW 2020

예제 4. 살짝 더 복잡한 서버 API 연동

- <AnimeList>는 비동기 데이터를 가져오는 animeList 셀렉터를 사용합니다.
- 개별 아이템은 stateless 컴포넌트인 <AnimeItem>이 렌더링합니다.

```
14  export default function AnimeList() {
15    const list = useRecoilValue(animeList);
16
17    return (
18      <div className="anime-list">
19        {list.map((item) => (
20          <AnimeItem key={item.mal_id} {...item} />
21        ))}
22      </div>
23    );
24  }
```

Learning by Examples

DEVIEW 2020

예제 4. 살짝 더 복잡한 서버 API 연동

- <AnimeList>를 <Suspense>로 감쌌습니다.

```
7   export default function App() {
8     return (
9       <div className="App">
10        <h1>애니메이션 검색</h1>
11        <p>영어 키워드를 입력한 후 엔터를 누르세요</p>
12        <RecoilRoot>
13          <SearchBox />
14          <Suspense fallback={<div>Loading...</div>}>
15            <AnimeList />
16          </Suspense>
17        </RecoilRoot>
18      </div>
19    );
20  }
```

Learning by Examples

DEVIEW 2020

예제 4. 살짝 더 복잡한 서버 API 연동

- state/index.js 파일에 아톰과 셀렉터를 각각 하나씩 선언했습니다.

```
1  import { atom, selector } from 'recoil';
2
3  export const keywordState = atom({
4    key: 'keywordState',
5    default: ''
6  });
7
8  export const animeList = selector({
9    key: 'animeList',
10   get: async ({ get }) => {
11     const keyword = get(keywordState);
12     if (!keyword || keyword.length < 3) {
13       return [];
14     }
15
16     const response = await fetch(
17       `https://api.jikan.moe/v3/search/anime?q=${keyword}&rated=pg13&page=1`
18     );
19     const data = await response.json();
20
21     return data.results;
22   }
23 });
```

Learning by Examples

DEVIEW 2020

예제 4. 살짝 더 복잡한 서버 API 연동

- state/index.js 파일에 아톰과 셀렉터를 각각 하나씩 선언했습니다.

```
3   export const keywordState = atom({
4     key: 'keywordState',
5     default: ''
6   });
7
8   export const animeList = selector({
9     key: 'animeList',
10    get: async ({ get }) => {
11      const keyword = get(keywordState);
12      if (!keyword || keyword.length < 3) {
13        return [];
14      }
```

사용한 아톰에 자동으로 의존성이 걸림

Lea

예제

- sta

DEVIEW 2020

Summary

DEVIEW 2020

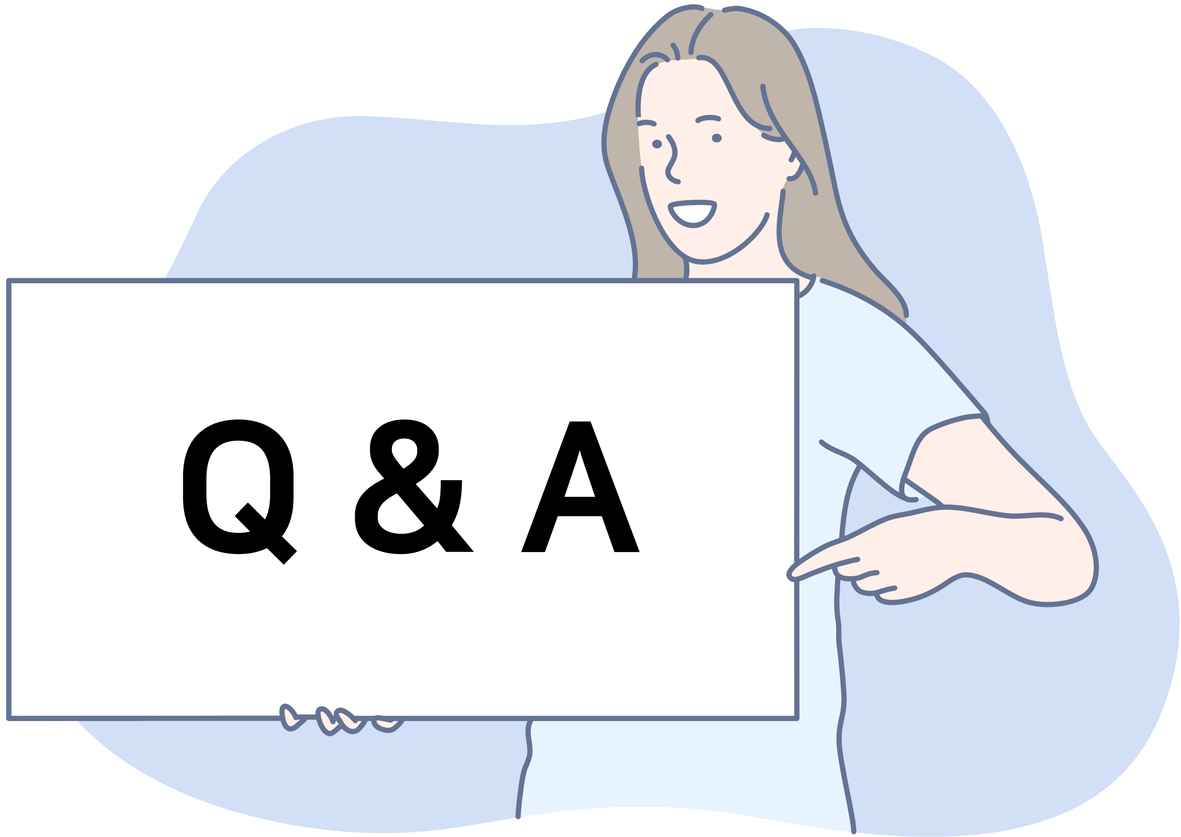
Recoil의 장점

- React와 굉장히 잘 연계됩니다.
- 사용법이 직관적이고 단순합니다.

Recoil의 단점

- Hooks를 통해서만 사용할 수 있습니다.
- 프로덕션 레벨에서 사용하기엔 아직 약간의 부담이 있습니다.
- 현재는 디버깅 도구의 지원이 미미합니다.

DEVIEW 2020



감사합니다