



**FEConf 2021**

안희종 (<https://ahnheejong.name>)

# 발표자 소개

- 임팩트에 집중하는 제품 엔지니어



# 발표자 소개



- 임팩트에 집중하는 제품 엔지니어
- 플렉스팀 프론트엔드 챕터 리드

# 발표자 소개



- 임팩트에 집중하는 제품 엔지니어
- 플렉스팀 프론트엔드 챕터 리드

## 새로운 HR의 시작

HR의 디지털 전환을 통한 업무 효율과 직원경험의 개선.  
flex와 함께라면 회사의 HR이 새로워집니다.

# 발표자 소개



- 임팩트에 집중하는 제품 엔지니어
- 플렉스팀 프론트엔드 챔터 리드

## 폭풍성장의 비결

빠르게 성장하는 회사는 flex를 사용합니다.  
flex를 쓰는 회사는 빠르게 성장합니다.

kakao

Melon

MIRAE ASSET

Hanwha

SK

FLO

서울특별시

KRIS

DEVSISTERS

NPIXEL

MOLOCO

idus

wadiz

ESTsoft

SANDBOX

ēlon

BALAAN

MATHPRESSO

peoplefund

gowid

신상마켓

Fitogther

CLASS101

knowre

VUNO

GENECAST

식권대장

Ti:mF

FRESHCODE

달인!

LEO

Lunit

DENTZ SECRET

ZERO BAKERY

kikike

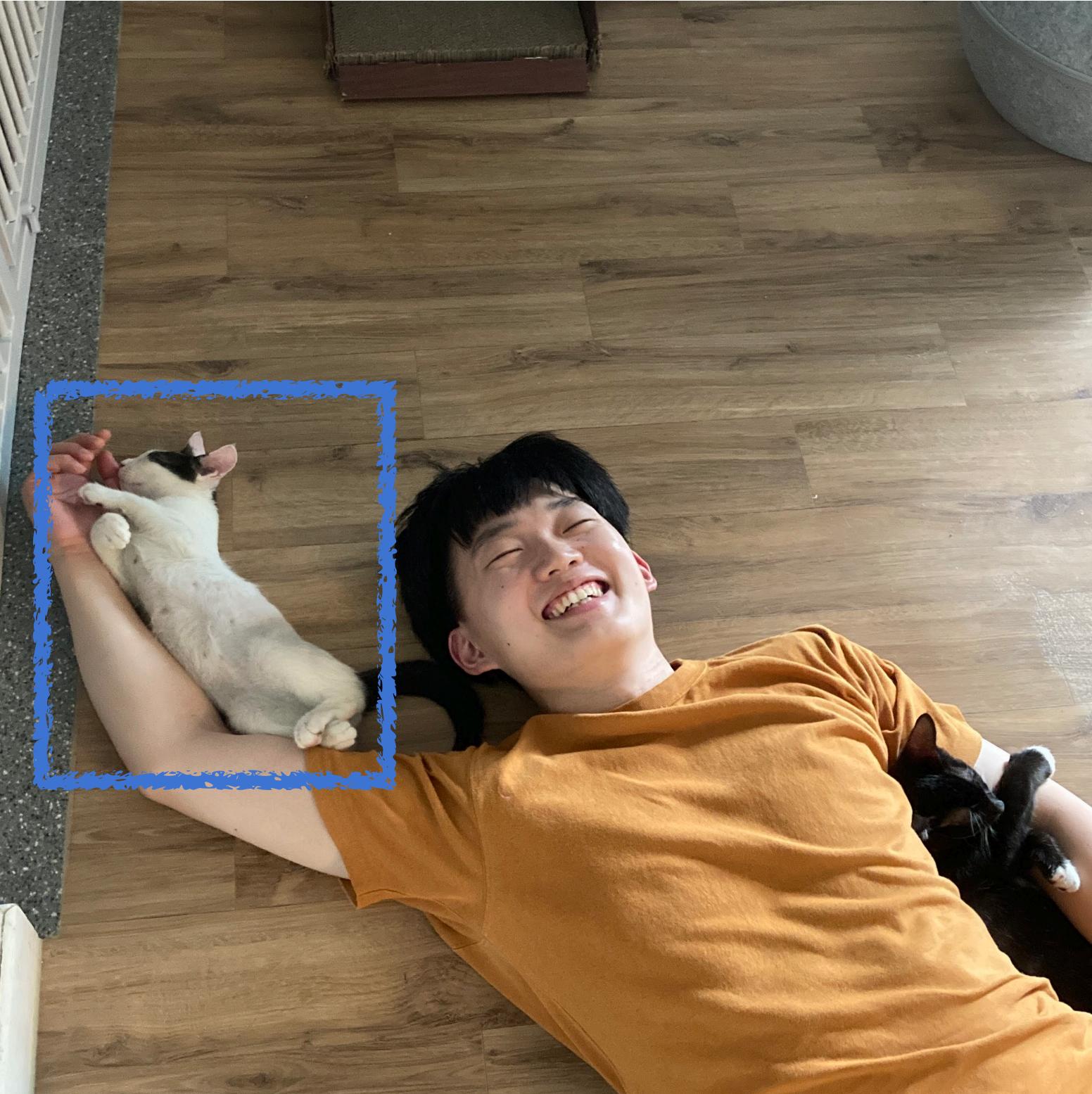
WISELY

# 발표자 소개



- 임팩트에 집중하는 제품 엔지니어
- 플렉스팀 프론트엔드 챕터 리드
- 두 고양이와 동거 중 (봄동, 달래)

# 발표자 소개



- 임팩트에 집중하는 제품 엔지니어
- 플렉스팀 프론트엔드 챕터 리드
- 두 고양이와 동거 중 (**봄동**, 달래)

# 발표자 소개



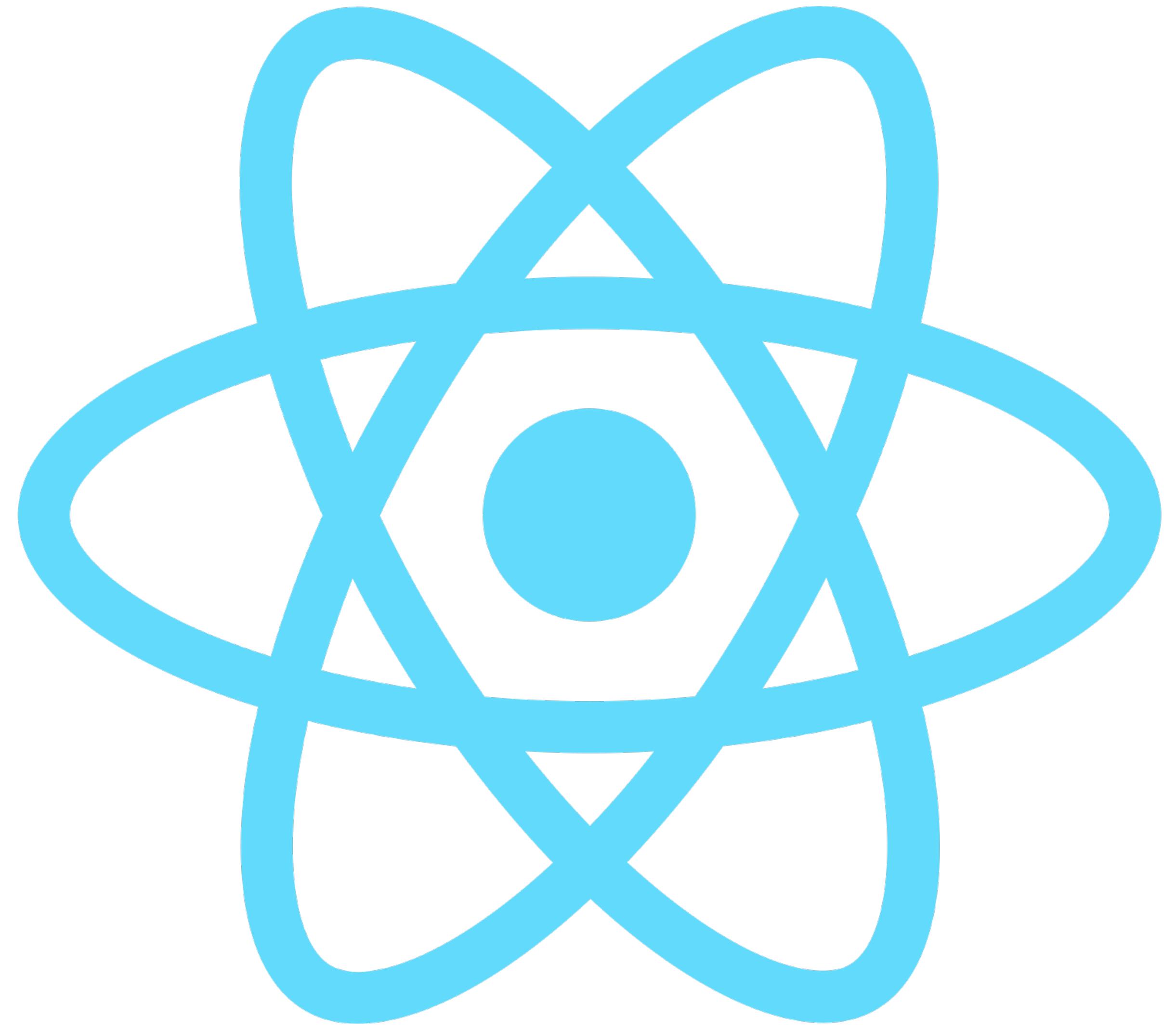
- 임팩트에 집중하는 제품 엔지니어
- 플렉스팀 프론트엔드 챕터 리드
- 두 고양이와 동거 중 (봄동, 달래)

# 발표자 소개



- 임팩트에 집중하는 제품 엔지니어
- 플렉스팀 프론트엔드 챕터 리드
- 두 고양이와 동거 중 (봄동, 달래)

**동기**



# **React 와 함께한 커리어**



# React 와 함께한 커리어



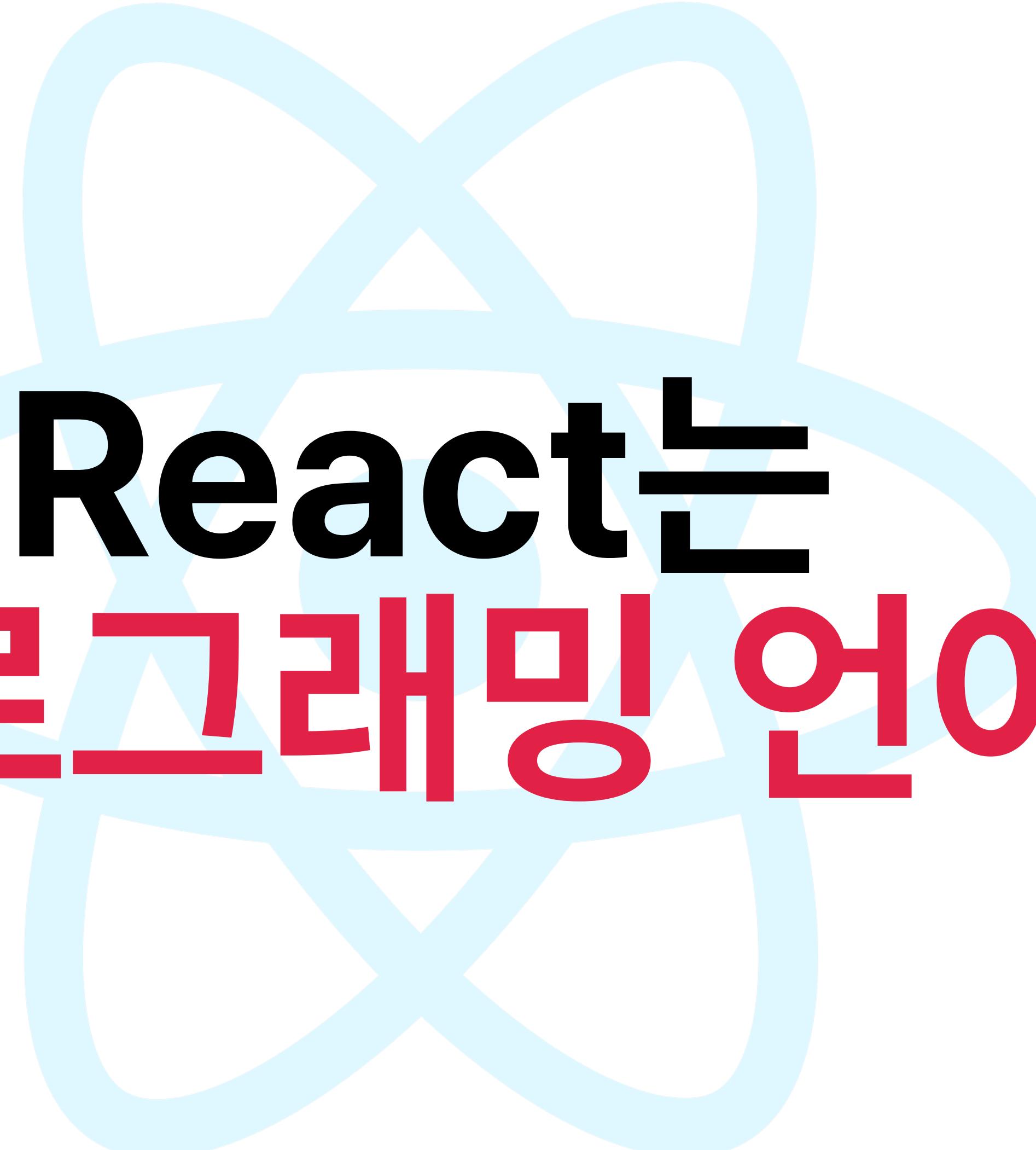
**왜 갈수록 더 좋아질까?**

# 왜 나는 널 사랑하는가

- 클리어한 멘탈 모델, 작고 단단한 코어
  - Learn Once, Write Anywhere
- 꾸준히 성장하는 거대한 커뮤니티
- 도전적인 과제, 우아한 해결책
  - Fiber, Hooks, Suspense, ...
  - “와, 어떻게 이런 생각을 했지?”

동기

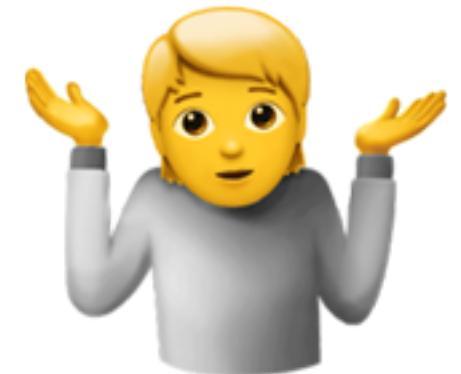
**React = 프로그래밍 언어?**



**React는  
UI 프로그래밍 언어다?**

# React

A JavaScript library for building user interfaces





# React와 프로그래밍 언어 : 값

```
var conferenceName = 'FECONF';
let year = 2021;
const isAwesome = true;
```

# React와 프로그래밍 언어 : 값

```
const element = (
  <h1 className='main-heading'>
    Hello, world
  </h1>
);
```

```
const elementWithoutJSX = React.createElement(
  'h1',
  { className: 'main-heading' },
  'Hello world',
);
```

# React와 프로그래밍 언어: 함수

```
function getQuestion() {  
  return `잘 지내시죠?`;  
}
```

```
function getSurvey(title) {  
  return {  
    title,  
    question: getQuestion(),  
  };  
}
```

# React와 프로그래밍 언어: 함수

```
function Question() {
  return (
    <div>
      잘 지내시죠?
    </div>
  );
}
```

```
function Survey({ title }: SurveyProps) {
  return (
    <div>
      <h1>{title}</h1>
      잘 지내시죠?
    </div>
  );
}
```

# React와 프로그래밍 언어 : 예외 처리

```
try {
  myFn();
} catch (e) {
  errorHandler(e);
}
```

# React와 프로그래밍 언어 : 예외 처리

```
class Try extends Component {
  constructor(props) {
    super(props);
    this.state = DEFAULT_STATE;
  }

  static getDerivedStateFromError(error) {
    return { hasError: true, error };
  }

  render() {
    if (this.state.hasError) {
      return this.props.catch({
        error: this.state.error,
      });
    }

    return this.props.children;
  }
}
```

```
<Try
  catch={({e})=><Fallback error={e}>/>
}>
  <Widget />
</Try>
```

# React와 프로그래밍 언어 : 타입 체크

```
let name: string;  
// '숫자' 값을 '문자열' 타입 변수에 할당할 수 없습니다.  
name = 42;
```

# React와 프로그래밍 언어 : 타입 체크

```
function HelloWorld({ name }) {
  return <div>Hello, {name}</div>;
}

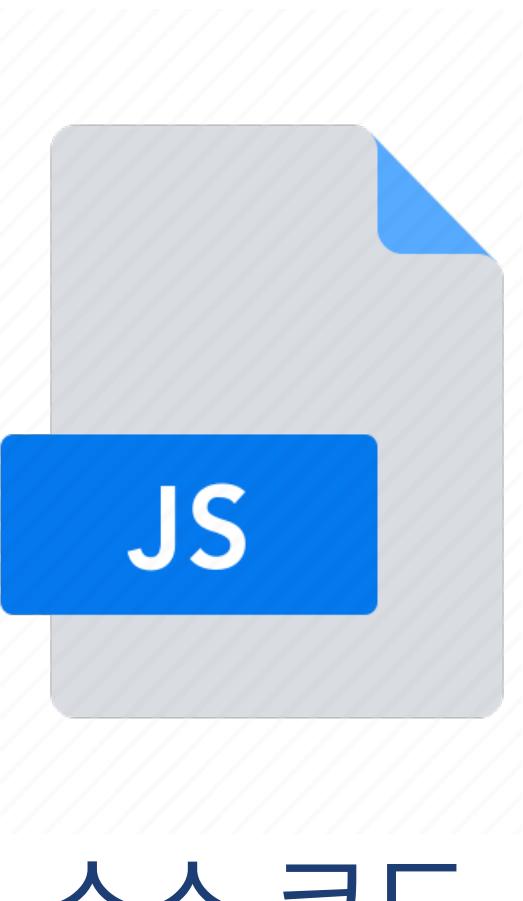
HelloWorld.propTypes = {
  name: PropTypes.string,
};

// name prop 은 올바르지 않습니다 – 문자열이여야 합니다.
<HelloWorld name={123} />;
```

# React와 프로그래밍 언어

프로그래밍 언어	리액트
값	React Element
함수	React Component
try-catch	Error Boundary
정적 타입 체크	prop-types

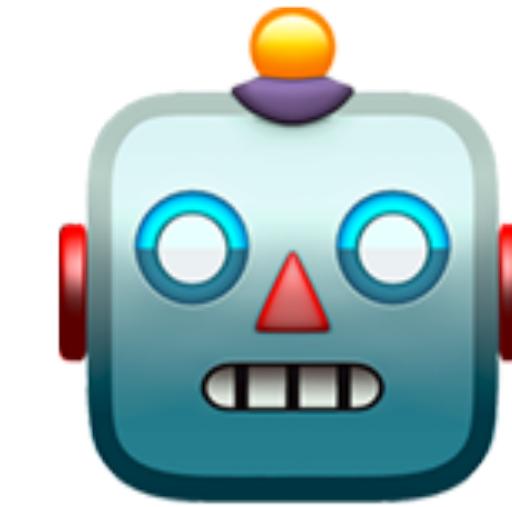
# **소스 코드, 엔진, 어셈블리**



소스 코드

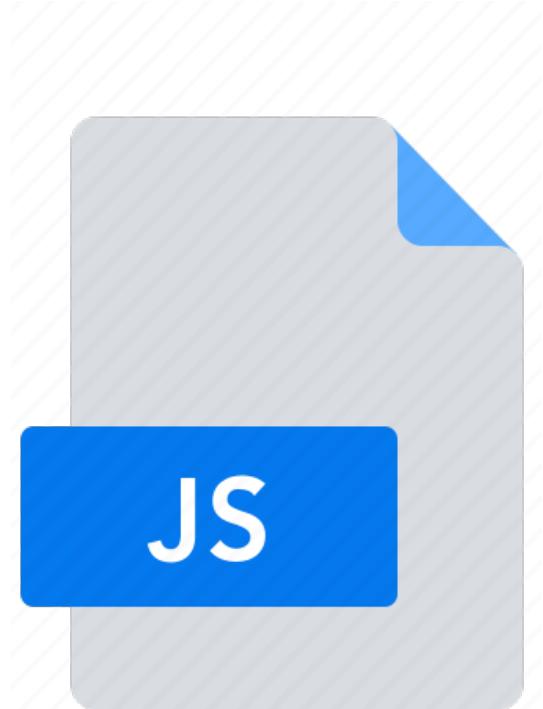


엔진



CPU

# 소스 코드, 엔진, 어셈블리

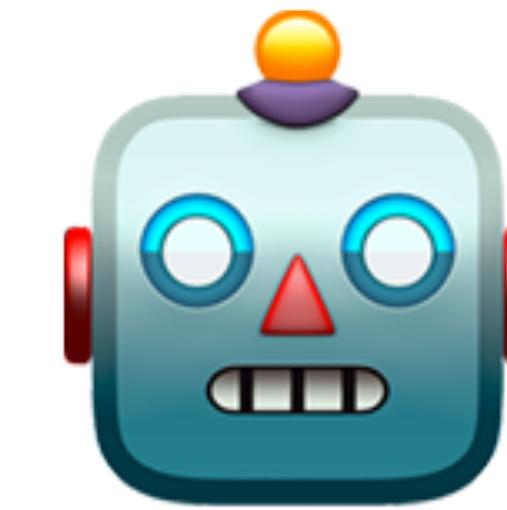
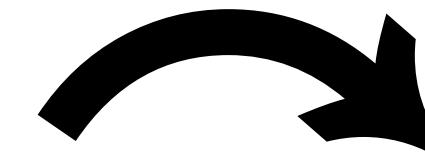


소스 코드



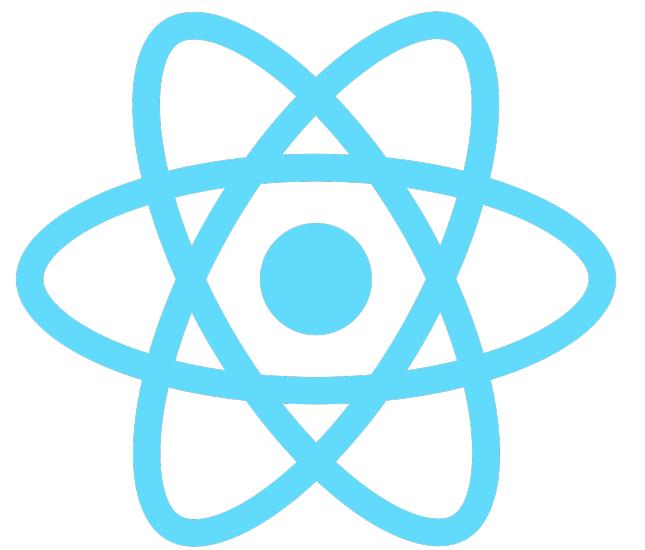
엔진

어셈블리



CPU

# React와 프로그래밍 언어 : 소스 코드, 엔진, 어셈블리



소스 코드

?

엔진

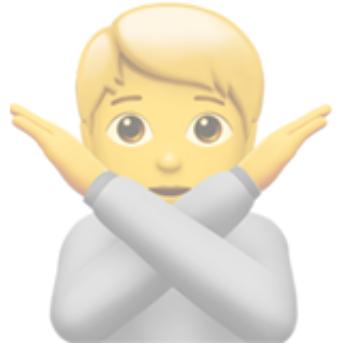
?

CPU

```
import React from "react";
import ReactDOM from "react-dom";
```



```
import React, { renderDOM } from "react";
```



The image is a composite of three elements. On the right side, there is a large React Conf 2019 logo featuring a stylized atom icon and the text "REACT CONF 2019". On the left side, there is a video feed of a woman with long dark hair, Sophie Alpert, speaking at a podium. She is wearing a patterned top and has a microphone attached to her shirt. In the center, there is a dark, semi-transparent window titled "talk.md" which contains a portion of a Markdown file. The file includes numbered lines of code and text, such as "host environments:" and "completely shared (reconciler)".

```
9
10
11 host environments:
12 * built-in components (div, span, img, etc)
13 * built-in components (View, Text, Image, etc)
14
15
16 completely shared (reconciler):
17 * function components
18 * class components
19 * props, state
20 * effects, lifecycles
21 * key, ref, context
22 * React.lazy, error boundaries
23 * concurrent mode, and Suspense
```

Sophie Alpert - Building a Custom React Renderer

A screenshot of a video conference interface. On the left, a woman with long dark hair, Sophie Alpert, is speaking at a podium with a laptop. She is wearing a patterned top. The background is a blue and yellow abstract design. On the right, there is a code editor window titled "talk.md". The code is as follows:

```
9  
10  
11 host environments:  
12 * built-in components (div, span, img, etc)  
13 * built-in components (View, Text, Image, etc)  
14  
15 completely shared (reconciler):  
16 * function components  
17 * class components  
18 * props, state  
19 * effects, lifecycles  
20 * key, ref, context  
21 * React.lazy, error boundaries  
22 * concurrent mode, and Suspense  
23
```

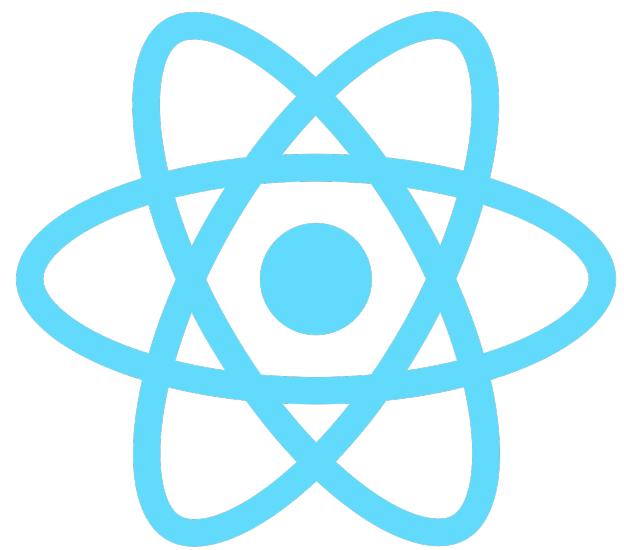
The text from line 15 to 23 is highlighted with a red box.

Below the code editor, the status bar shows: Ln 11, Col 19, Spaces: 2, UTF-8, LF, Markdown, Prettier, a smiley face icon, and a bell icon.

At the bottom right, the React Conf 2019 logo is displayed, featuring a stylized atom icon and the text "REACT CONF 2019".

Sophie Alpert - Building a Custom React Renderer

# React와 프로그래밍 언어 : 소스 코드, 엔진, 어셈블리



소스 코드

reconciler

엔진

?

CPU

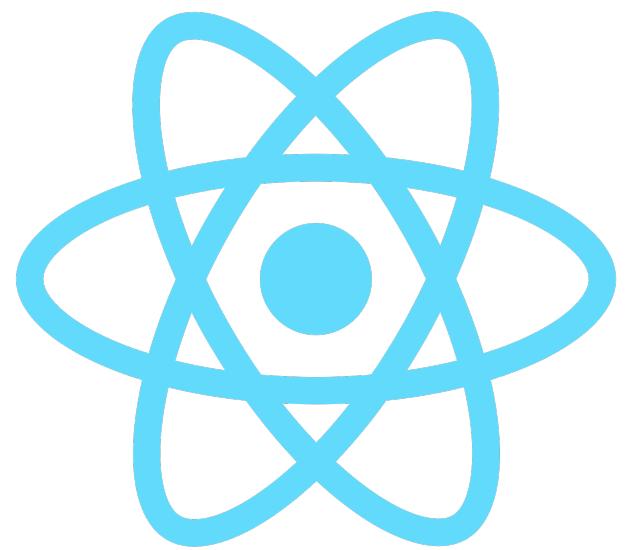
talk.md

```
9
10
11 host environments:
12 * built-in components (div, span, img, etc)
13 * built-in components (View, Text, Image, etc)
14
15 completely shared (reconciler):
16 * function components
17 * class components
18 * props, state
19 * effects, lifecycles
20 * key, ref, context
21 * React.lazy, error boundaries
22 * concurrent mode, and Suspense
```

호스트 환경  
= UI가 렌더되는 “기계”

호스트 환경과의 상호작용 수단  
= 어셈블리

# React와 프로그래밍 언어 : 소스 코드, 엔진, 어셈블리



소스 코드

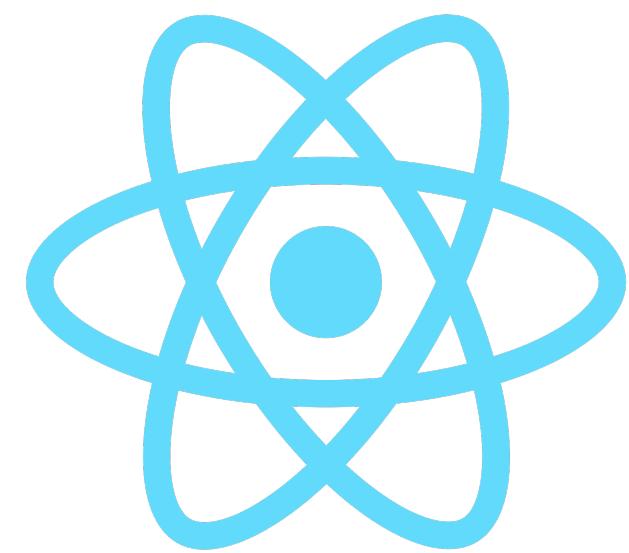
reconciler

엔진

호스트  
환경

CPU

# React와 프로그래밍 언어 : 소스 코드, 엔진, 어셈블리



소스 코드

reconciler

엔진

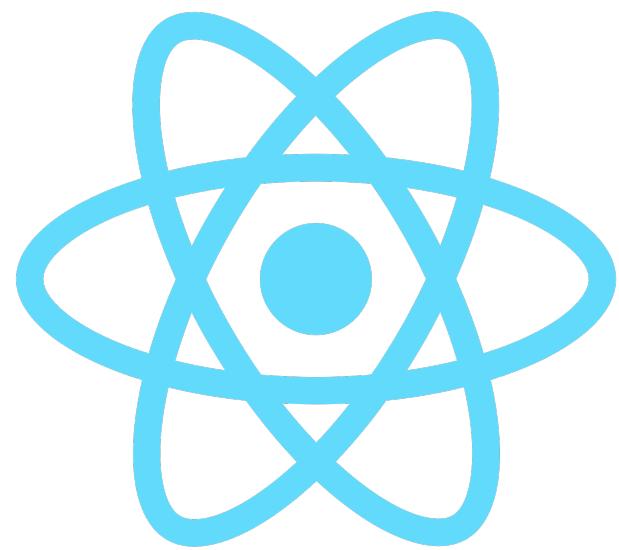
새로운 노드 추가



호스트  
환경

CPU

# React와 프로그래밍 언어 : 소스 코드, 엔진, 어셈블리



소스 코드

reconciler

엔진



<임의의\_호스트\_환경을\_조작하는\_수단>

Array.prototype.push

document.appendChild

<임의의\_호스트\_환경>

JSON

HTML

호스팅  
환경

CPU

# Learn Once, Write Anywhere

호스팅 환경	패키지
HTML	react-dom
iOS / Android / ...	react-native
터미널	ink
Three.js	react-three-fiber
JSON	react-test-renderer

# Learn Once, Write Anywhere

호스팅 환경

패키지

## 다른 호스트 환경 (어셈블리) 동일한 React API (문법)

Three.js

react-three-fiber

JSON

react-test-renderer

# React와 프로그래밍 언어

프로그래밍 언어	리액트
값	React Element
함수	React Component
try-catch	Error Boundary
정적 타입 체크	prop-types
엔진	Reconciler
어셈블리	호스트 환경

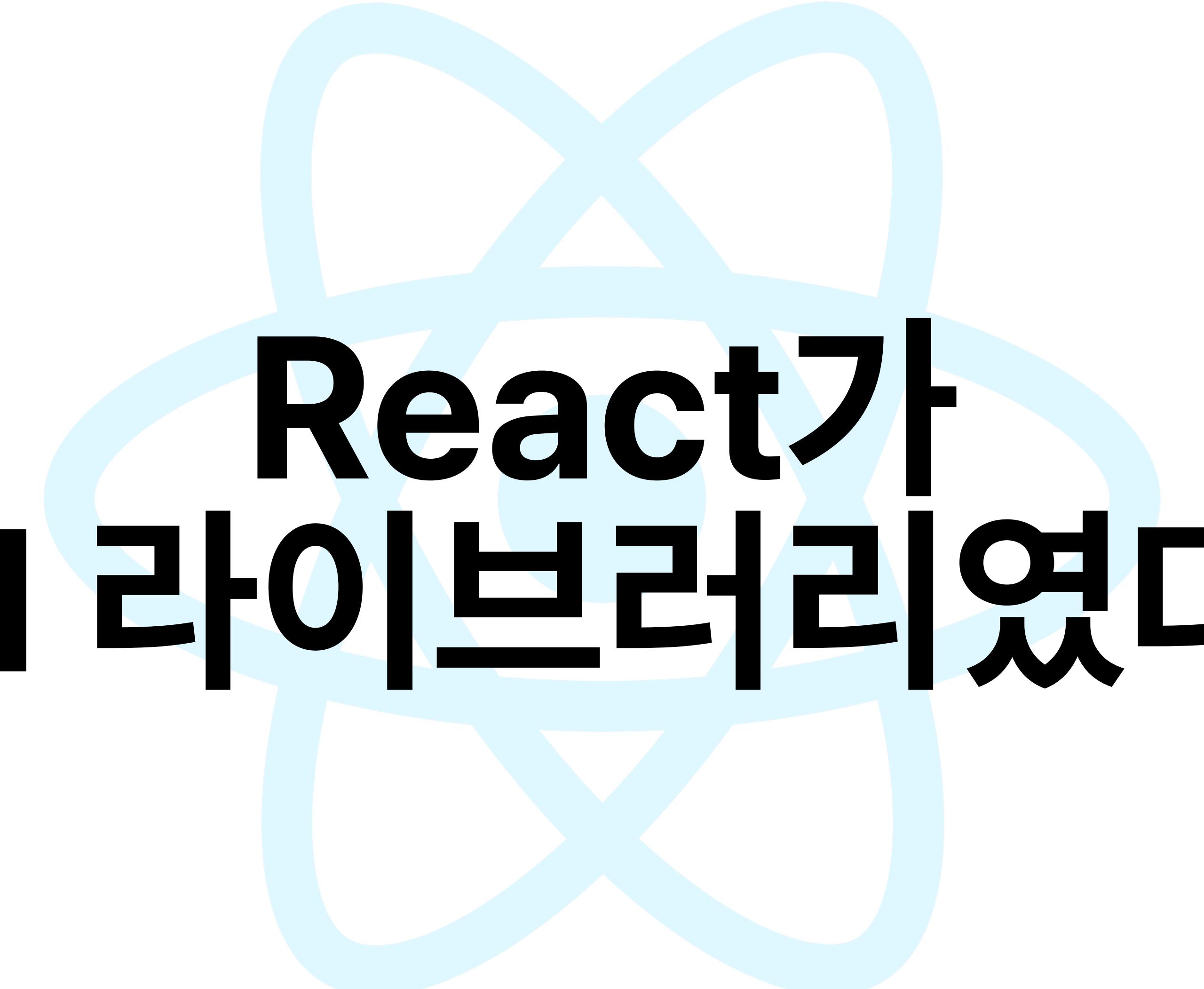
# 왜 나는 널 사랑하는가

- 클리어한 멘탈 모델, 작고 단단한 코어
  - Learn Once, Write Anywhere
- 꾸준히 성장하는 거대한 커뮤니티
- 도전적인 과제, 우아한 해결책
  - Fiber, Hooks, Suspense, ...
  - “와, 어떻게 이런 생각을 했지?”

동기

React = 프로그래밍 언어?

# 명확한 문제 및 목표 정의



**웹 UI 라이브러리였다면?**

React가

# 어쩌면...

```
import React from "react";
import ReactDOM from "react-dom";
```



```
import React, { renderDOM } from "react";
```

# 어쩌면...

호스팅 환경	패키지
HTML	react
iOS / Android / ..	react-native
JSON	react-test-renderer

**“Virtual DOM은  
너무 느려”**

# **Just JavaScript**

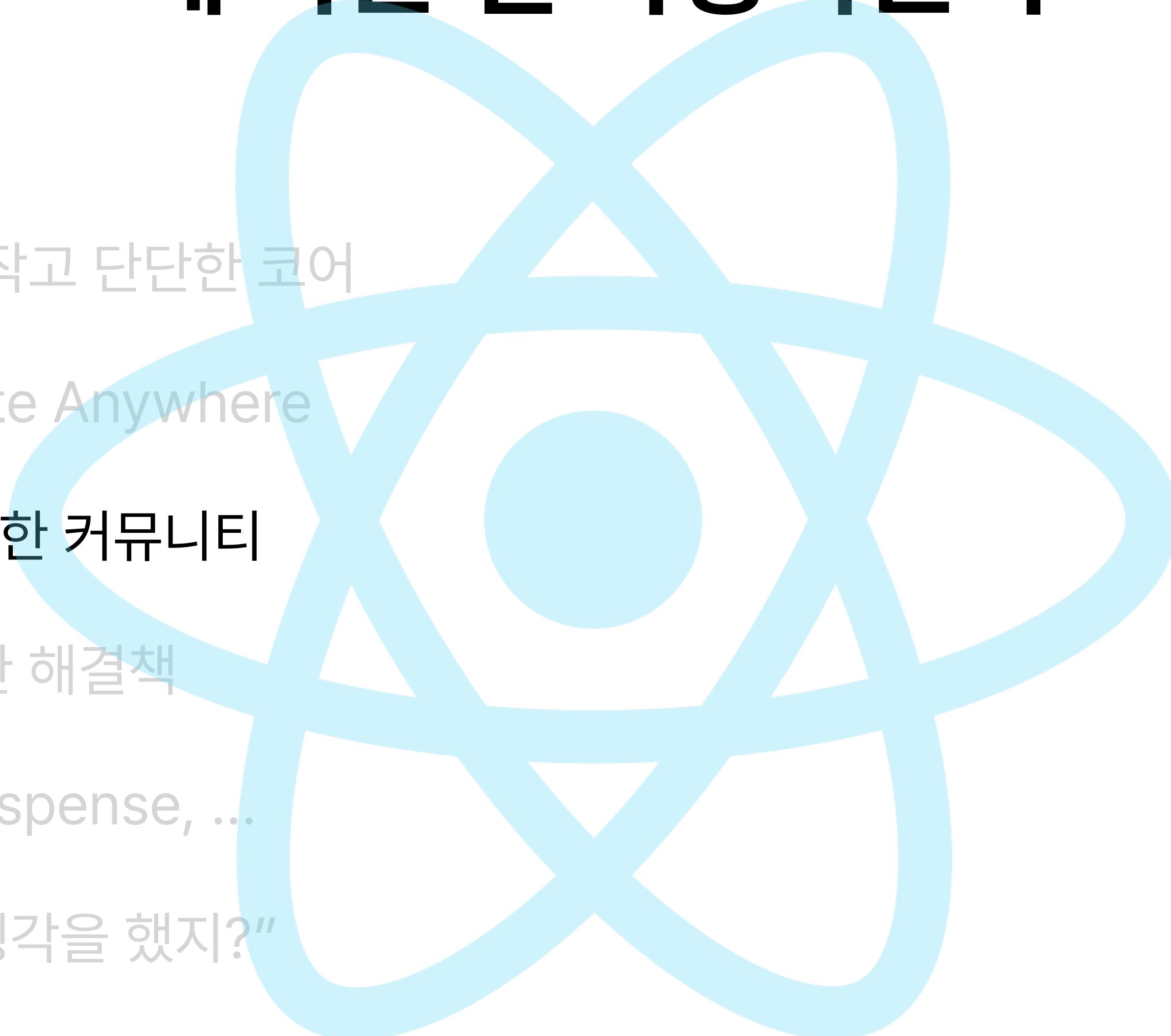
**Just JavaScript**  
**≠ 브라우저 런타임**

**Just JavaScript  
≠ Node.js**

# **Just JavaScript**

# 왜 나는 널 사랑하는가

- 클리어한 멘탈 모델, 작고 단단한 코어
  - Learn Once, Write Anywhere
- 꾸준히 성장하는 거대한 커뮤니티
- 도전적인 과제, 우아한 해결책
  - Fiber, Hooks, Suspense, ...
  - “와, 어떻게 이런 생각을 했지?”



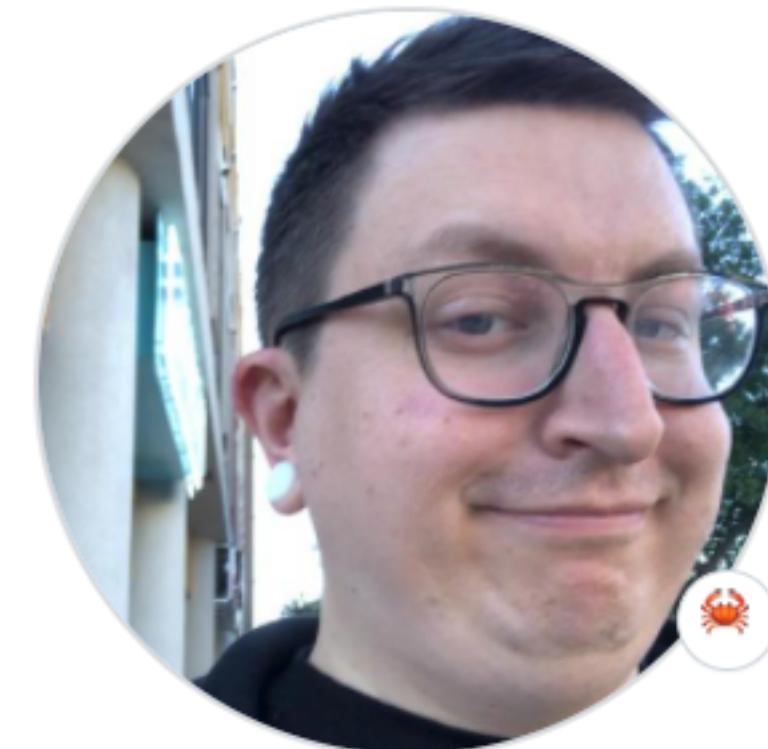
동기

React = 프로그래밍 언어?

명확한 문제 및 목표 정의

# 낯섦 예산의 현명한 소비

# The language strangeness budget



**Steve Klabnik**  
[steveklabnik](https://steveklabnik.com)

프로그래밍 언어를 만드는 건 쉽지만 사람들이  
그 언어를 쓰게 만드는 건 훨씬, 훨씬 어렵습니다. (...)  
당신의 언어에 새롭고, 재미있고, 흥분되는 기능이  
얼마나 많이 포함되어 있는지 인지하고, 그 수를 세심  
하게 고려해야 합니다.

When it comes to programming languages, building one  
is easy, but getting people to use it is much, much harder.  
(...) you need to be aware of how many new, interesting,  
exciting things that your language is doing, and carefully  
consider the number of such features you include.

프로그래밍 언어를 만드는 건 쉽지만 사람들이  
그 언어를 쓰게 만드는 건 훨씬, 훨씬 어렵습니다. (...)  
당신의 언어에 새롭고, 재미있고, 흥분되는 기능이  
얼마나 많이 포함되어 있는지 인지하고, 그 수를 세심  
하게 고려해야 합니다.

When it comes to programming languages, building one  
is easy, but getting people to use it is much, much harder.  
(...) you need to be aware of how many new, interesting,  
exciting things that your language is doing, and carefully  
consider the number of such features you include.

가치 > 비용

&&

전체 예산 > 전체 비용



Elm



ReScript

# What's A Language

- ▶ Community
- ▶ Libraries
- ▶ Package Management
- ▶ Compiler

Syntax → TypeSystem → Backend

# What's A Language

- ▶ Community
- ▶ Libraries
- ▶ Package Management
- ▶ Compiler

Syntax → TypeSystem → Backend

# 리액트가 언어였다면...

부족한 리소스,  
텅 빈 Stack Overflow

JSX

새로운  
패키지 매니저

컴포넌트 기반  
개발

선언적  
렌더링

작은 유저 풀

# 리액트가 언어였다면...

부족한 리소스,  
텅 빈 Stack Overflow

**JSX**

새로운  
패키지 매니저

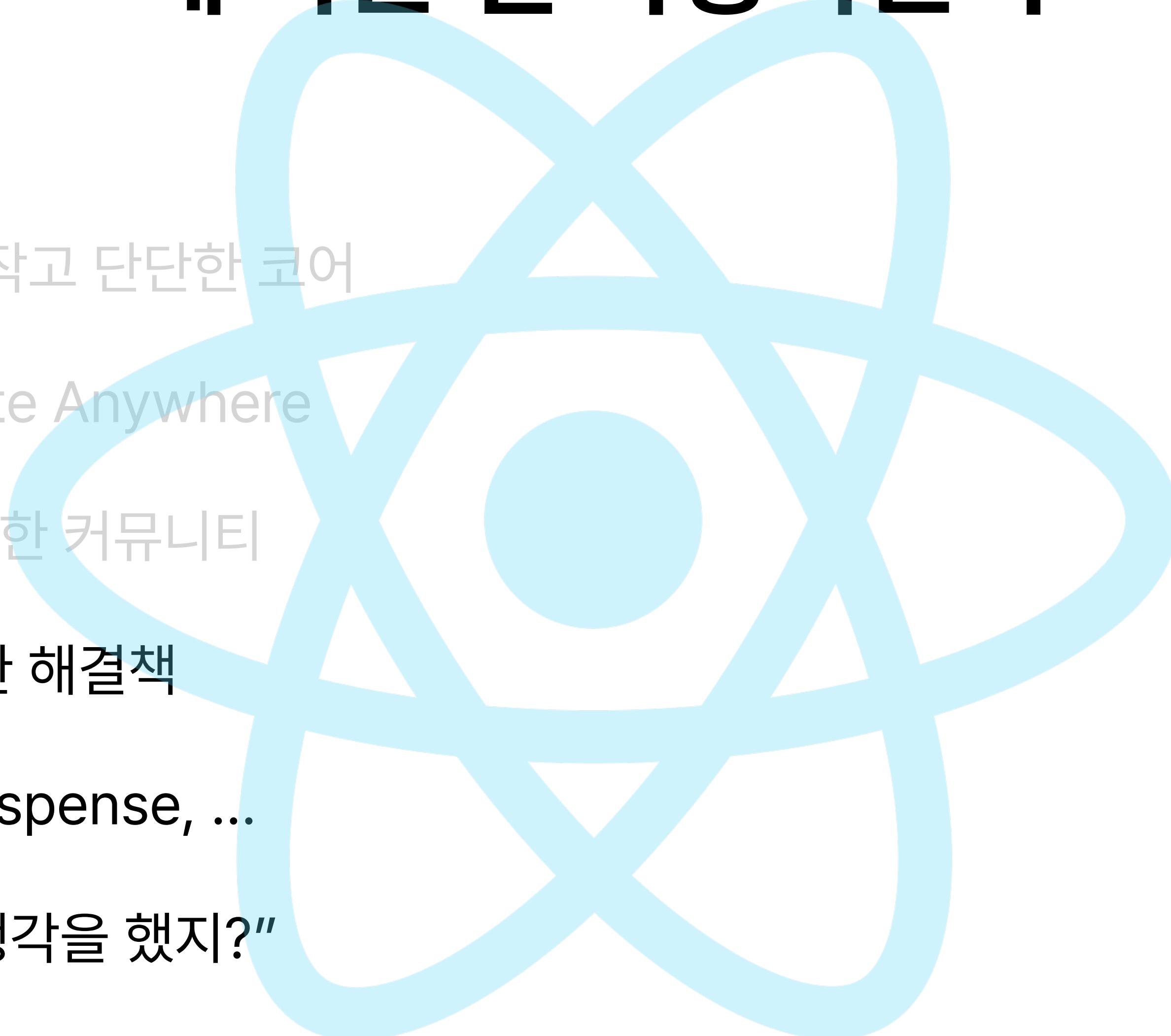
컴포넌트 기반  
개발

선언적  
렌더링

작은 유저 풀

# 왜 나는 널 사랑하는가

- 클리어한 멘탈 모델, 작고 단단한 코어
  - Learn Once, Write Anywhere
- 꾸준히 성장하는 거대한 커뮤니티
- 도전적인 과제, 우아한 해결책
  - Fiber, Hooks, Suspense, ...
  - “와, 어떻게 이런 생각을 했지?”



동기

React = 프로그래밍 언어?

명확한 문제 및 목표 정의

낮은 예산의 현명한 소비

# 답안지 훔쳐보기



Blitzscaling 11: Patrick Collison on Hiring at Stripe and the Role of a Product-Focused CEO

Q. 당신은 제가 트위터에서 팔로우하는 사람 중 온갖 것의 역사에 대해 읽는 데 가장 많은 시간을 쓰는 사람 중 한 명인데요. 그러는 이유가 있나요?

Q. You spend more time reading history of everything, really, than almost anybody I follow on Twitter. Why do you do that?

## A. 편법이죠. 많은 사람이 역사 속 좋은 아이디어를 무시해요 (...)

엄청나게 고유한 생각을 떠올리기 위해 가만히 앉아서 며칠 내내 벽을 바라보며 노력할 수도 있지만, 편법을 쓸 수도 있어요 (...) 그런 아이디어는 말 그대로 책에 쓰여 있어서, 그냥 읽으면 됩니다.

A. It's a way to cheat. Everyone else ignores all the good ideas from history (...) you could just try to think all these incredibly original thoughts by sort of sitting down and staring at the wall for days on end, or you could sort of cheat (...) they are, in fact, written down in books that you can just read.

A. 편법이죠. 많은 사람이 역사 속 좋은 아이디어를 무시해요 (...)  
엄청나게 고유한 사고를 떠올리기 위해 가만히 앉아서 며칠 내내  
벽을 바라보며 노력할 수도 있지만, 편법을 쓸 수도 있어요 (...)  
그런 아이디어는 말 그대로 책에 쓰여 있어서, 그냥 읽으면 됩니다.

A. It's a way to cheat. Everyone else ignores all the good ideas from history (...) you could just try to think all these incredibly original thoughts by sort of sitting down and staring at the wall for days on end, or you could sort of cheat (...) they are, in fact, written down in books that you can just read.

A. 편법이죠. 많은 사람이 역사 속 좋은 아이디어를 무시해요 (...)  
엄청나게 고유한 생각을 떠올리려 가만히 앉아서 며칠 내내 벽을  
바라보며 노력할 수도 있지만, 일종의 편법을 쓸 수도 있어요 (...)  
그런 아이디어는 말 그대로 책에 쓰여 있어서, 그냥 읽으면 됩니다.

A. It's a way to cheat. Everyone else ignores all the good ideas from history (...) you could just try to think all these incredibly original thoughts by sort of sitting down and staring at the wall for days on end, or you could sort of cheat (...) they are, in fact, written down in books that you can just read.



Dan @dan\_abramov · 6월 10일

@dan\_abramov 님에게 보내는 답글

I felt in the dark many times. I felt we've let down the community by sharing our vision too early and now having to focus on putting out the fires while people think we forgot or don't care. There's one thing @sebmarkbage told me many years ago that kept me going.

1

3

53



Dan @dan\_abramov · 6월 10일

"Our work is rooted in theory. In theory, some approach is better. So we pursue it. Many people don't trust the theory. But we keep on going. While it may take many practical compromises and pragmatic sacrifices to get there. If we keep on going, eventually the theory wins."

1

22

158



[https://twitter.com/dan\\_abramov/status/1402943255759798272](https://twitter.com/dan_abramov/status/1402943255759798272)



Sebastian Markbåge  
sebmarkbage

Dan @dan\_abramov · 6월 10일  
@dan\_abramov 님에게 보내는 답글

I felt in the dark many times. I felt we've let down the community by sharing our vision too early and now having to focus on putting out the fires while people think we forgot or don't care. There's one thing @sebmarkbage told me many years ago that kept me going.

1 3 53

Dan @dan\_abramov · 6월 10일

“Our work is rooted in theory. In theory, some approach is better. So we pursue it. Many people don't trust the theory. But we keep on going. While it may take many practical compromises and pragmatic sacrifices to get there. If we keep on going, eventually the theory wins.”

1 22 158

우리의 작업물은 이론에 뿌리를 둡니다.  
이론 상 더 나은 접근은 분명 존재합니다.  
따라서 우리는 그 방향을 추구합니다.

많은 사람들이 이론을 신뢰하지 않습니다.  
그렇지만 우리는 멈추지 않습니다.  
목적지에 도달하기 위해 수많은 실질적 탐험과  
실용적인 희생을 치르더라도 말이죠.

우리가 멈추지 않는다면, 결국 이론이 이깁니다.

우리의 작업물은 이론에 뿌리를 둡니다.  
이론 상 더 나은 접근은 분명 존재합니다.  
따라서 우리는 그 방향을 추구합니다.

많은 사람들이 이론을 신뢰하지 않습니다.  
그렇지만 우리는 멈추지 않습니다.  
목적지에 도달하기 위해 수많은 실질적 타협과  
실용적인 희생을 치르더라도 말이죠.

우리가 멈추지 않는다면, 결국 이론이 이깁니다.

우리의 작업물은 이론에 뿌리를 둡니다.  
이론 상 더 나은 접근은 분명 존재합니다.  
따라서 우리는 그 방향을 추구합니다.

Dan @dan\_abramov · 6월 10일  
많은 사람들이 이론을 신뢰하지 않습니다.  
그렇지만 우리는 멈추지 않습니다.  
목적지에 도달하기 위해 수많은 실질적 타협과  
실용적인 희생을 치르더라도 말이죠.

우리가 멈추지 않는다면, 결국 이론이 이깁니다.

**환원 (Reduction)**

# 환원

새로운 문제 A 가 생겼는데

이러이러한 면에서

답안지가 존재하는 문제 B 랑 되게 비슷한데?!

# 환경 : Context

컴포넌트가 “트리 저 위쪽에 정의된 무언가”로부터  
값을 꺼내올 수단이 필요한데…

참조 시점에 둘러싼 환경을 기준으로 평가되는 변수니까

프로그래밍 언어의 동적 스코핑이랑 되게 비슷한데?

# 환경 : Context

컴포넌트가  
값을 꺼내올  
참조 시점에

## Context

In React, we pass things down to other components as props. Sometimes, the majority of components need the same thing — for example, the currently chosen visual theme. It gets cumbersome to pass it down through every level.

In React, this is solved by [Context](#). It is essentially like [dynamic scoping](#) for components. It's like a wormhole that lets you put something on the top, and have every child at the bottom be able to read it and re-render when it changes.

<https://overreacted.io/react-as-a-ui-runtime/>

프로그래밍 언어의 동적 스코핑이랑 되게 비슷한데?

# 환원 : Fiber Reconciler

JavaScript의 싱글 스레드 환경에서  
느린 컴포넌트가 우선순위 높은 업데이트를 막아 생기는  
반응성 저해를 방지할 수단이 필요한데…

하나의 실행 주체로 우선순위가 다른 여러 작업이  
동시에 잘 실행되게 만드는 일이니까

운영체제의 스케줄링이랑 되게 비슷한데?!

# 환원 : Hooks

함수 컴포넌트의 제약을 제거하고  
상태 로직의 응집성·재사용성을 개선<sup>하고 싶은데…</sup>

특정 “효과” (상태, 라이프사이클 이펙트, …) 의 처리를  
React에게 위임할 수단을 찾아야 하는 <sup>거니까</sup>

대수적 효과<sup>랑 되게 비슷한데?!</sup>

# 환원 : Suspense

비동기로 불러와야 하는 리소스를

선언적으로 정의<sup>하고 싶은데…</sup>

특정 “효과” (비동기 호출) 의 처리를

React에게 위임할 수단을 찾아야 하는 거니까

대수적 효과<sup>랑 되게 비슷한데?!</sup>

# React의 환원

문제	환원된 문제	답안지	결과물
특정 트리의 모든 컴포넌트가 공유하는 값을 정의할 수단	참조하는 시점에 둘러싼 환경을 기준으로 평가되는 값	동적 스코핑	Context
싱글 스레드 환경 내 렌더링이 느린 컴포넌트가 UI 반응성을 저해하는 상황 방지	하나의 실행 주체로 서로 우선순위가 다른 여러 작업이 동시에 실행되는 듯 보이게 하기	운영 체제의 스케줄링 비선점형 스케줄리의 Fiber	Fiber Reconciler
함수 컴포넌트의 제약 제거, 상태 로직의 응집성·재사용성 증대	특정 '효과' (상태, 라이프사이클, 비동기, ...)를 React가 제공한 핸들러에 위임할 수단	대수적 효과	Hooks
비동기 데이터 수집 과정의 선언적 처리			Suspense

동기

React = 프로그래밍 언어?

명확한 문제 및 목표 정의

낮은 예산의 현명한 소비

답안지 훔쳐보기

# 사례 : 플렉스팀

# 리모트 레이어

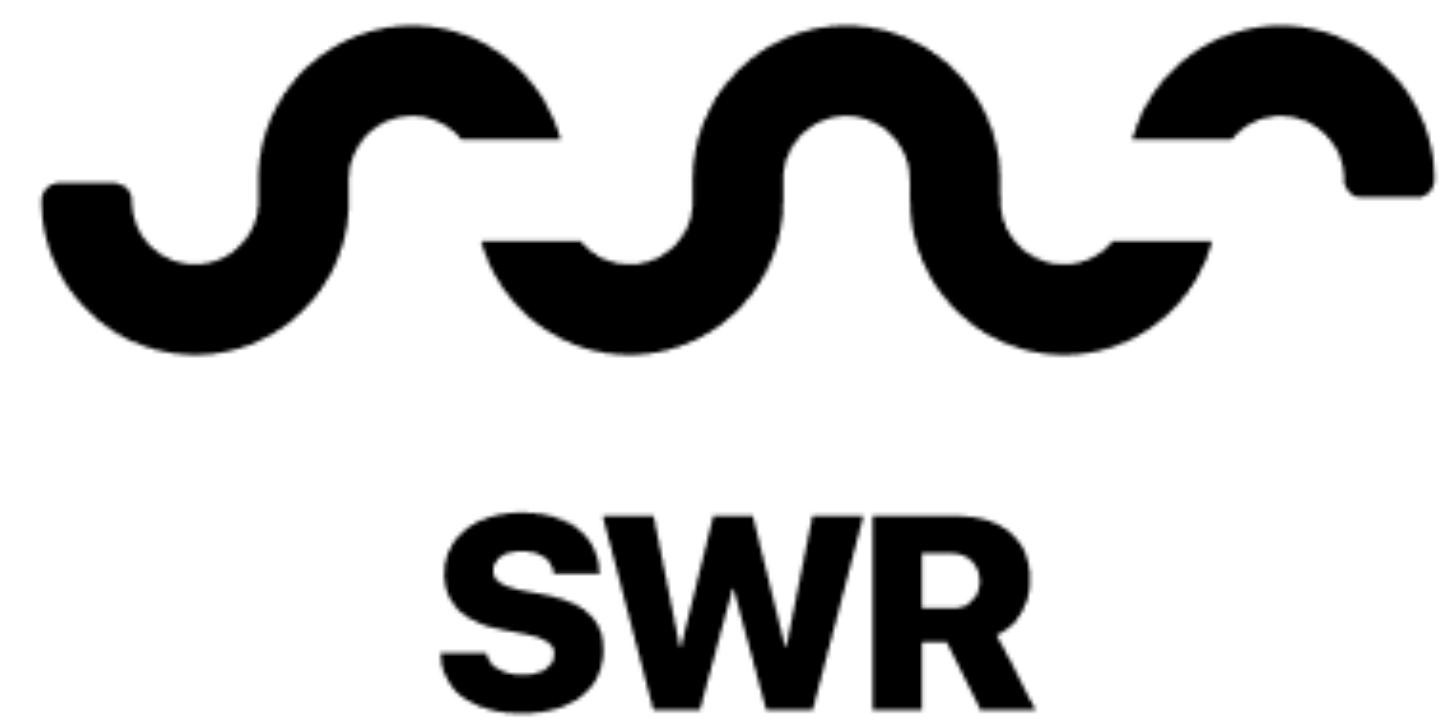
서버 API 모델 정의, 엔드포인트 호출, 데이터 관리, ...

```
async function getUserProfile(id: string) {  
  const data = await axios.get<Response>(  
    `/action/v1/user-profile?id=${id}`  
  );  
  return data;  
}
```

```
async function getUserProfile(id: string) {  
  const data = await axios.get<Response>(  
    `/action/v1/user-profile?id=${id}`  
  );  
  return data.data;  
}
```

```
async function getUserProfile(id: string) {  
  const data = await axios.get<Response>(  
    `/action/v1/user-profile`,  
    {  
      params: { id },  
    }  
  );  
  return data.data;  
}
```

```
async function getUserProfile(  
  id: string,  
  axiosInstance: AxiosInstance = axios  
) {  
  const data = await axiosInstance.get<Response>(  
    `/action/v1/user-profile`,  
    {  
      params: { id },  
    }  
  );  
  return data.data;  
}
```



React Hooks for Remote Data Fetching

```
function useUserProfile(id: string) {
  const key = `userProfile?userId=${id}`;
  return useSWR(
    key,
    () => getUserProfile(id)
  );
}
```

```
function useUserProfile(id: string) {
  const key = `@flex-apps/user-profile/hooks/useUserProfile/${id}`;
  return useSWR(
    key,
    () => getUserProfile(id)
  );
}
```

```
function useUserProfile(id: string) {
  const key = [`@flex-apps/user-profile/hooks/useUserProfile`, id];
  return useSWR(
    key,
    () => getUserProfile(id)
  );
}
```

```
function useUserProfile(id: string) {
  const key = [`@flex-apps/user-profile/hooks/useUserProfile`, id];
  return useSWR(
    key,
    () => getUserProfile({ id })
  );
}
```

**리모트 레이어를  
사랑스럽게  
만들 수 없을까?**

# 명확한 문제 및 목표 정의

# 문제?

리모트 리소스 정의 방식,  
swr 키 생성 방식이  
사용처 별로 제각각이다?

목표?

리모트 레이어 사용  
컨벤션 마련?

# 문제

리모트 리소스 정의,  
swr 키 생성 등의 작업을  
개발자가 수동으로 반복하고 있다

# 목표

리모트 레이어

사용 컨벤션 마련

자동화!

낯설 예산의  
현명한 소비

# 낯설 예산의 현명한 소비

(feat. 할부)

# 1. 리모트 리소스 정의 방식 일원화

```
const getUserProfileRemote = makeRemote<Response, Params>(
  {
    method: 'get',
    url: '/action/v1/user-profile',
  },
  id => {
    return {
      params: { id },
    };
  }
);

const [getUserProfile] = getUserProfileRemote(axios);
getUserProfile('join@flex.team');
```

1. 리모트 리소스 정의 방식 일원화

2. 리모트 리소스 → 리모트 푸

```
import {  
  getUserProfileRemote,  
} from '../remotes';  
  
const useUserProfile = (id: string) => {  
  return useQueryFromRemote({  
    remote: getUserProfileRemote,  
    extraParams: { id },  
  });  
};
```

1. 리모트 리소스 정의 방식 일원화
2. 리모트 리소스 → 리모트 푸

 마일스톤 1:  
서버 호출 함수 일원화  
일원화된 리소스로부터 푸 생성 자동화

1. 리모트 리소스 정의 방식 일원화
2. 리모트 리소스 → 리모트 푸



1. 리모트 리소스 정의 방식 일원화
2. 리모트 리소스 → 리모트 헥
3. 스펙 → 리모트 리소스

```
// OpenAPIGenerator 를 사용해 자동 생성
import {
  getUserProfileRemote,
} from '@flex-apis/user-profile';

const useUserProfile = (id: string) => {
  return useQueryFromRemote({
    remote: getUserProfileRemote,
    extraParams: { id },
  });
};
```

1. 리모트 리소스 정의 방식 일원화
2. 리모트 리소스 → 리모트 푸
3. 스펙 → 리모트 리소스

 마일스톤 2:  
서버 호출 함수 생성까지 자동화

**답안지 흰쳐보기**

안희종 2.0 3개월 전

SWR 공식 예시에 비슷한 컨셉이 있네요.

<https://github.com/vercel/swr/blob/master/examples/axios-typescript/libs/useRequest.ts>

쓸 때는 <https://github.com/vercel/swr/blob/master/examples/axios-typescript/pages/%5Buser%5D/%5Brepo%5D.tsx> 이런 식으로 쓰고, 사용자는 key라는 개념을 아예 안 생각해도 됩니다.

```
const { data } = useRequest<{
  forks_count: number
  stargazers_count: number
  watchers: number
}>({
  url: '/api/data',
  params: { id }
})
```

JSON.stringify 가 serializer로 충분할지는 확인이 필요할 것 같네요

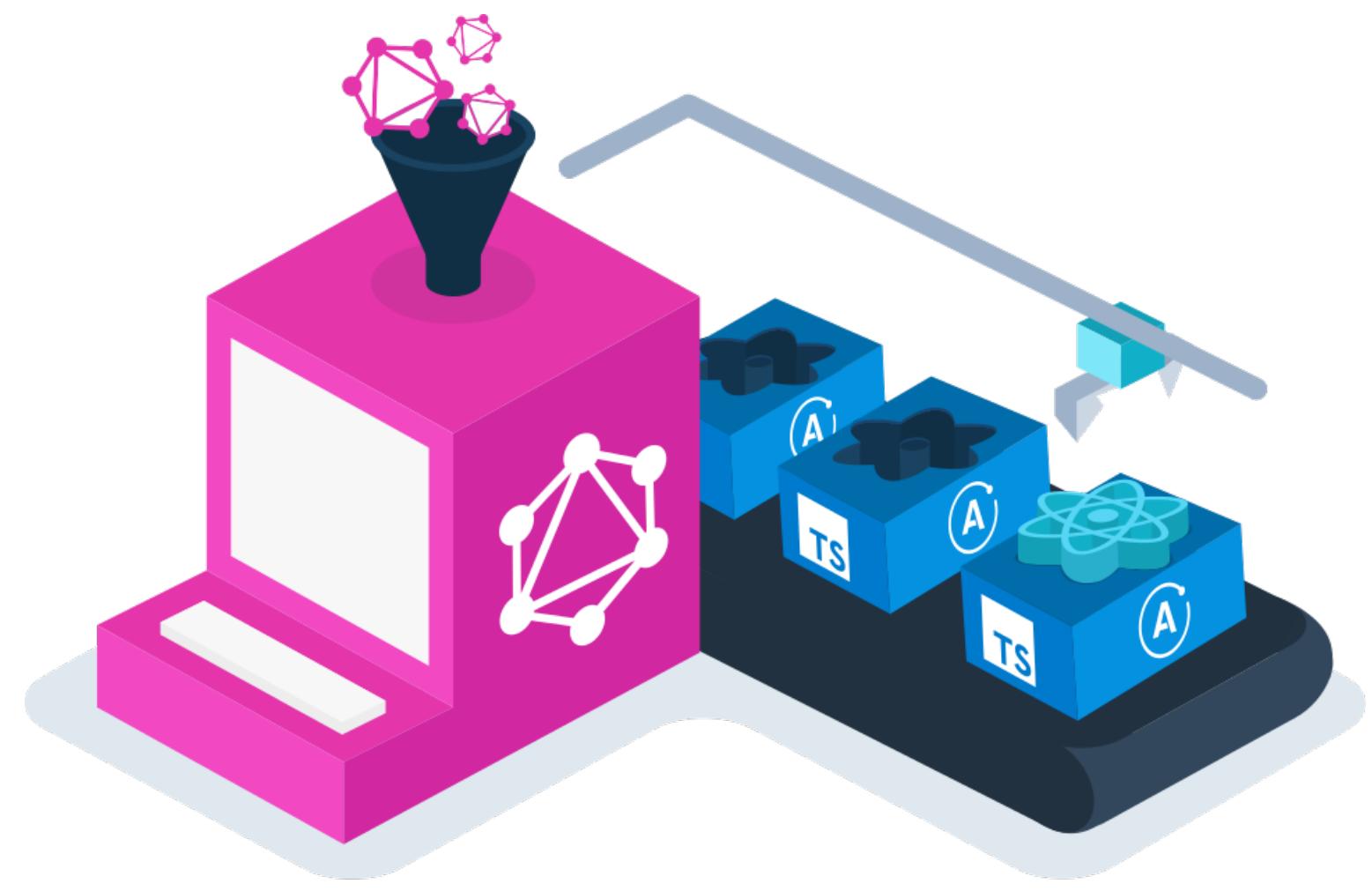
[useRequest.ts](#)

 vercel/swr | 봇이 추가한 GitHub

[\[repo\].tsx](#)

 vercel/swr | 봇이 추가한 GitHub

```
1 import useSWR, { SWRConfiguration, SWRResponse } from 'swr'
2 import axios, { AxiosRequestConfig, AxiosResponse, AxiosError } from 'axios'
3
4 export type GetRequest = AxiosRequestConfig | null
5
6 interface Return<Data, Error>
7   extends Pick<
8     SWRResponse<AxiosResponse<Data>, AxiosError<Error>>,
9     'isValidating' | 'error' | 'mutate'
10  > {
11    data: Data | undefined
12    response: AxiosResponse<Data> | undefined
13  }
14
15 export interface Config<Data = unknown, Error = unknown>
16   extends Omit<
17     SWRConfiguration<AxiosResponse<Data>, AxiosError<Error>>,
18     'fallbackData'
19   > {
20    fallbackData?: Data
21  }
--
```



{ GraphQL }  
**code generator**

graphql-code-generator



openapi-generator



A presentation slide from FE CONF 2020. The slide has a black background with abstract red and purple circular graphics on the right. In the top left corner, there's a small circular profile picture of a man with glasses. To the right of the profile picture, the text reads "OpenAPI Specification으로 타입 세이프하게 API 개발하기 - 희망편 vs 절망편". At the bottom right, it says "최태건 @MESH KOREA". The top right corner features the text "B TRACK". In the top left corner, there's a logo with the text "FE CONF 2020".

**<https://recruiting.flex.team>**

# 사례 : 플렉스팀

- 명확한 문제 및 목표 정의
  - 리모트 레이어의 자동화
- 낮설 예산의 현명한 소비
  - 너무 비싼 가격이 우려될 땐 할부 수단 제공
- 답안지 훔쳐보기
  - useRequest, graphql-code-generator, openapi-generator, FEConf 발표

동기

React = 프로그래밍 언어?

명확한 문제 및 목표 정의

낮은 예산의 현명한 소비

답안지 훔쳐보기

사례 : 플렉스팀

**결론**

# 왜 나는 리액트를 사랑하는가

- 많은 이유가 있지만, 결국 **프로그래밍 언어를 닮았다는** 지점으로 모아짐
- 매력 포인트와 교훈
  - 클리어한 멘탈 모델, Learn Once, Write Everywhere → **명확한 문제 및 목표 정의**
  - 꾸준히 성장하는 거대한 커뮤니티 → **낮은 예산의 현명한 소비**
  - 도전적인 과제, 우아한 해결책 → **답안지 훔쳐보기**
- 이러한 교훈은 일상 업무에도 충분히 적용 가능

# 참고 자료

- Dan Abramov - React as a UI Runtime
- Steve Klabnik - The language strangeness budget
- Sophie Alpert - Building a Custom React Renderer
- Brandon Dail - Algebraic effects, Fibers, Coroutines...Oh my!
- Jordan Walke - React to the Future
- Blitzscaling 11: Patrick Collison on Hiring at Stripe and the Role of a Product-Focused CEO

# 왜 나는 리액트를 사랑하는가

- 많은 이유가 있지만, 결국 **프로그래밍 언어를 닮았다는 지점으로 모아짐**
- 매력 포인트와 교훈
  - 클리어한 멘탈 모델, Learn Once, Write Everywhere → **명확한 문제 및 목표 정의**
  - 꾸준히 성장하는 거대한 커뮤니티 → **낮쉼 예산의 현명한 소비**
  - 도전적인 과제, 우아한 해결책 → **답안지 훔쳐보기**
- 이러한 교훈은 일상 업무에도 충분히 적용 가능
- 우리 코드도 리액트만큼 사랑받는 그 날까지 !
- 감사합니다