

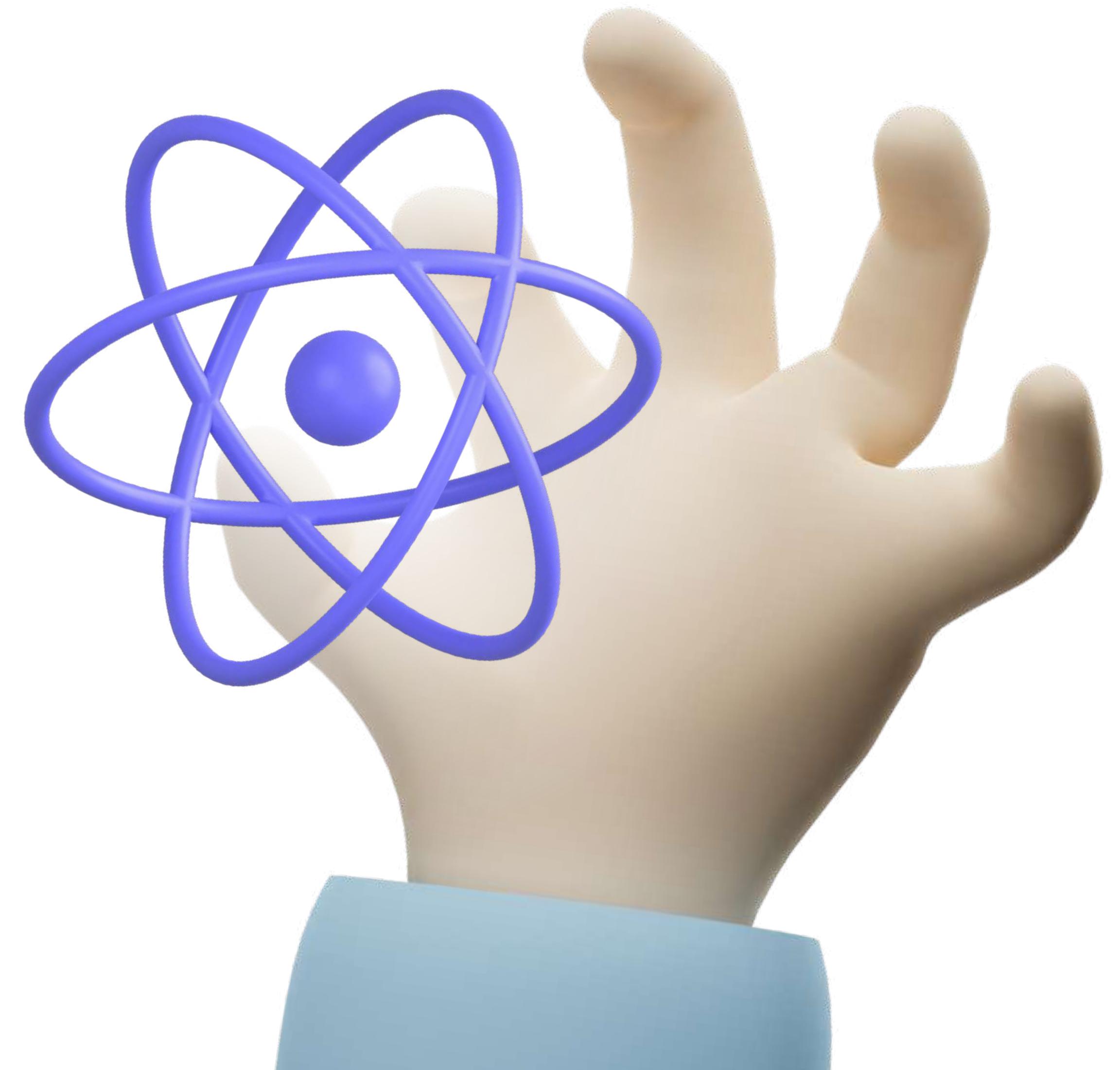


fe:ssion

likelion frontend week03

lead. junkyu lee

- 01 과제 점검 + 회고 + 우당탕탕 깃허브 대소동
- 02 React + jsx + CRA
- 03 기본 구조
- 04 함수형 컴포넌트
- 05 Component Driven Development 실습
- 06 hooks (useState, useEffect, useRef)와 렌더링
- 07 실습 + 과제 공지





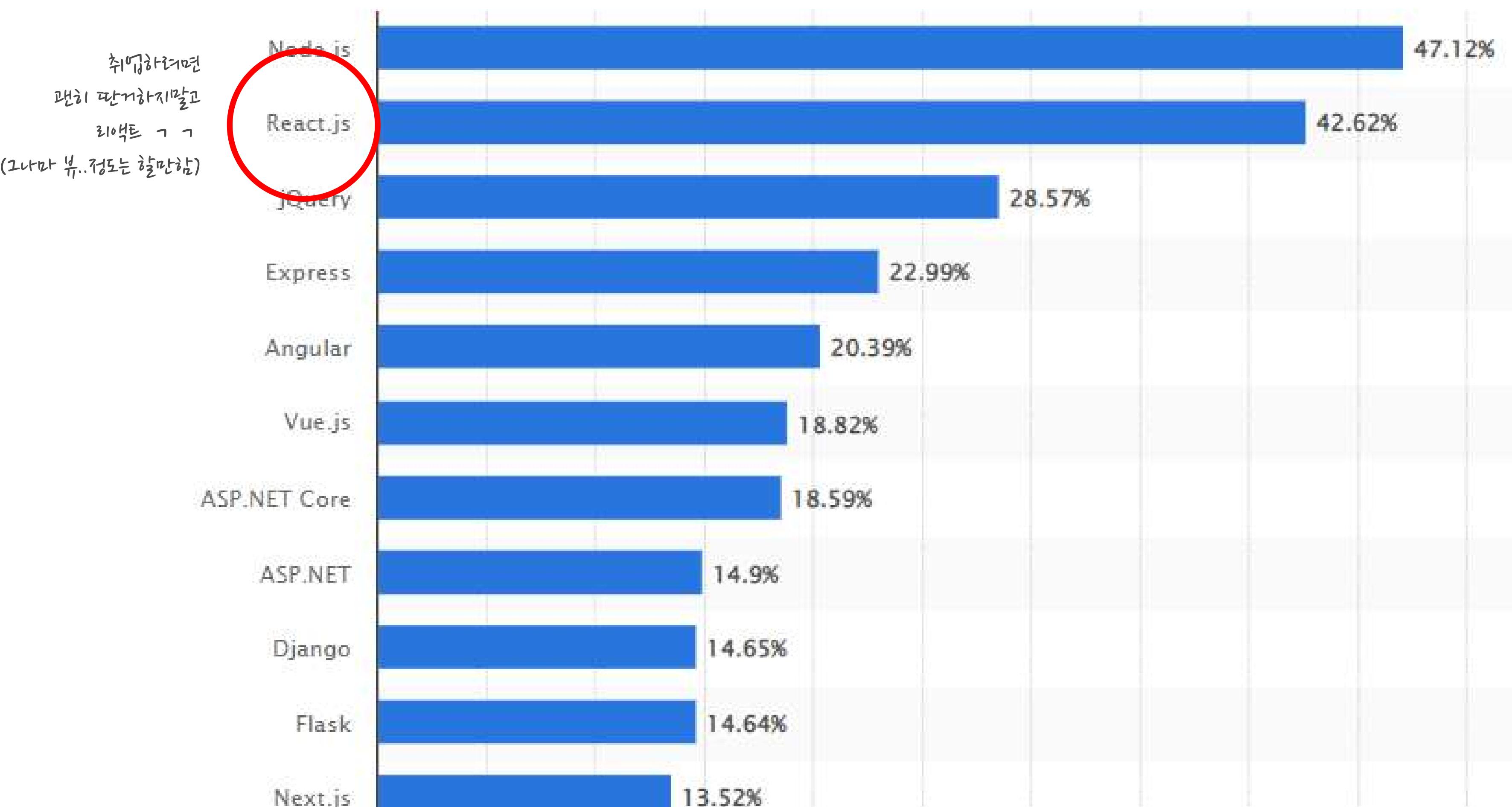
과제... 잘 하셨나요?

많이 어려웠나요? 힘들었나요? 할만했나요? 어땠나요?

이 페이지 또 보는 것 같다면 착각임 ㄹㅇ임

Meta가 만든 웹 라이브러리

WEB Application을 만들 때 개발 경험을 극도로 편하게 만들어주는, [User Interface](#)를 제작하기 위한 자바스크립트 라이브러리



SPA(Single Page Application)

모던 웹 패러다임.

단일 페이지 내부에서 여러가지 DOM이 생성되고 사라지며 웹 사이트를 구성함.

우리가 흔히 사용하는 [모바일 어플리케이션](#)처럼 작동하는 특징이 있음!



JSON(JavaScript Object Notation)이란,
데이터를 저장하거나 전송할 때 쓰는 경량의 데이터 교환 형식.
자바스크립트에서 객체를 만들 때 사용하는 표현식을 의미함

전통적인 MPA와 다른 점

무겁게 생각하지말고, 웹이 어플리케이션 같아졌다고 보면 됨!

SPA도 분명한 단점이 존재하기에, 최근에는 MPA의 장점과 SPA의 장점을 합치려는 시도가 이어지고 있음.

CSR

빠른 인터랙션

한 페이지만 불러오는 SPA 방식으로,
필요한 부분만 리로딩 없이 간신할 수 있게됨

트래픽 감소

필요한, 변경된 데이터만 받아오기 때문

사용자 경험

새로고침이 발생하지 않아
사용자가 네이티브 앱과 같은 경험을 할 수 있게됨

단점

초기 구동 속도 느림
검색엔진 최적화(SEO) 어려움

항목	MPA	SPA
페이지 수	여러 개	한 개
초기 구동 속도	느림	빠름
트래픽 용량	작음	큼
SEO 최적화	약함	강함
사용자 경험	자연스러움	깜박거림 (새로고침)
새로운 페이지 요청	필요한 부분 다운 / 렌더	전체 페이지 다운 / 렌더
렌더링 방식	클라이언트 사이드 렌더링 (CSR)	서버 사이드 렌더링 (SSR)

SSR

검색엔진 최적화(SEO) 가능

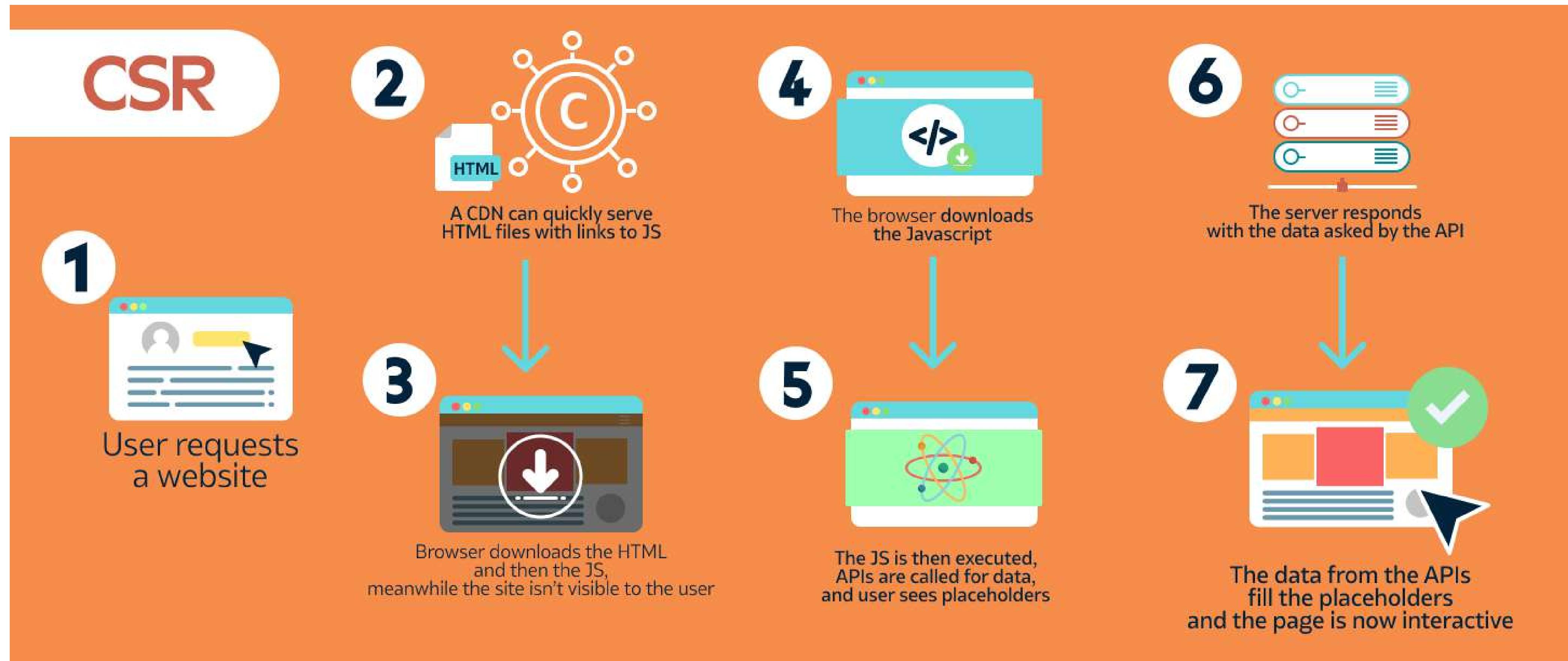
이미 렌더링된 HTML이 배포되기 때문에,
검색엔진(Robot)이 내부 콘텐츠를 인식하기가 쉬움

초기 로딩 속도 빠름

이미 렌더링된 HTML을 받기 때문에

단점

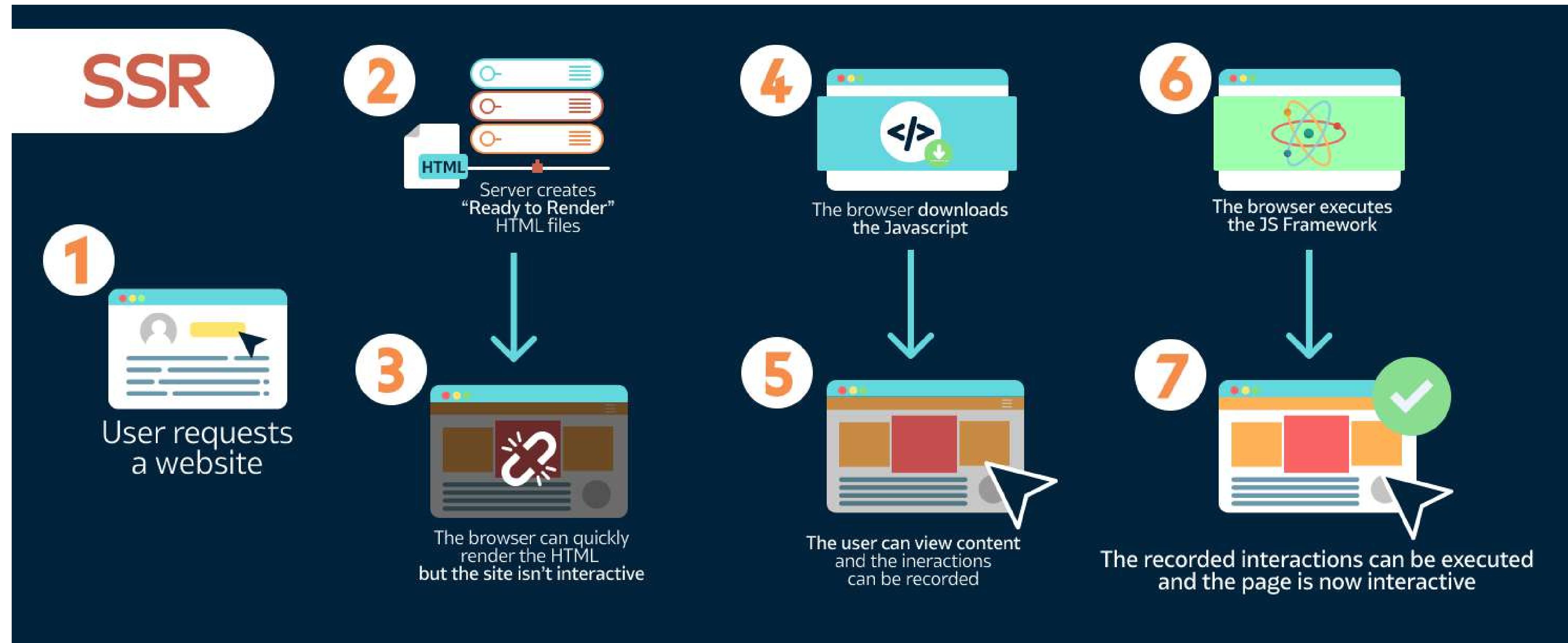
페이지 이동시 화면 깜빡임
서버 렌더링에 따른 부하 발생
페이지 요청마다 새로고침 발생



사용자와 상호작용하는 웹 어플리케이션

웹 앱의 경우 사용자와의 상호작용이 주.

때문에 각 반응이 오래걸리지 않는 CSR을 사용하는 것이 좋다



정보를 제공하는 웹 사이트

검색엔진에서 정보를 제공하는 웹 사이트의 경우
올바른 메타데이터를 보장하는 SSR이 좋다.

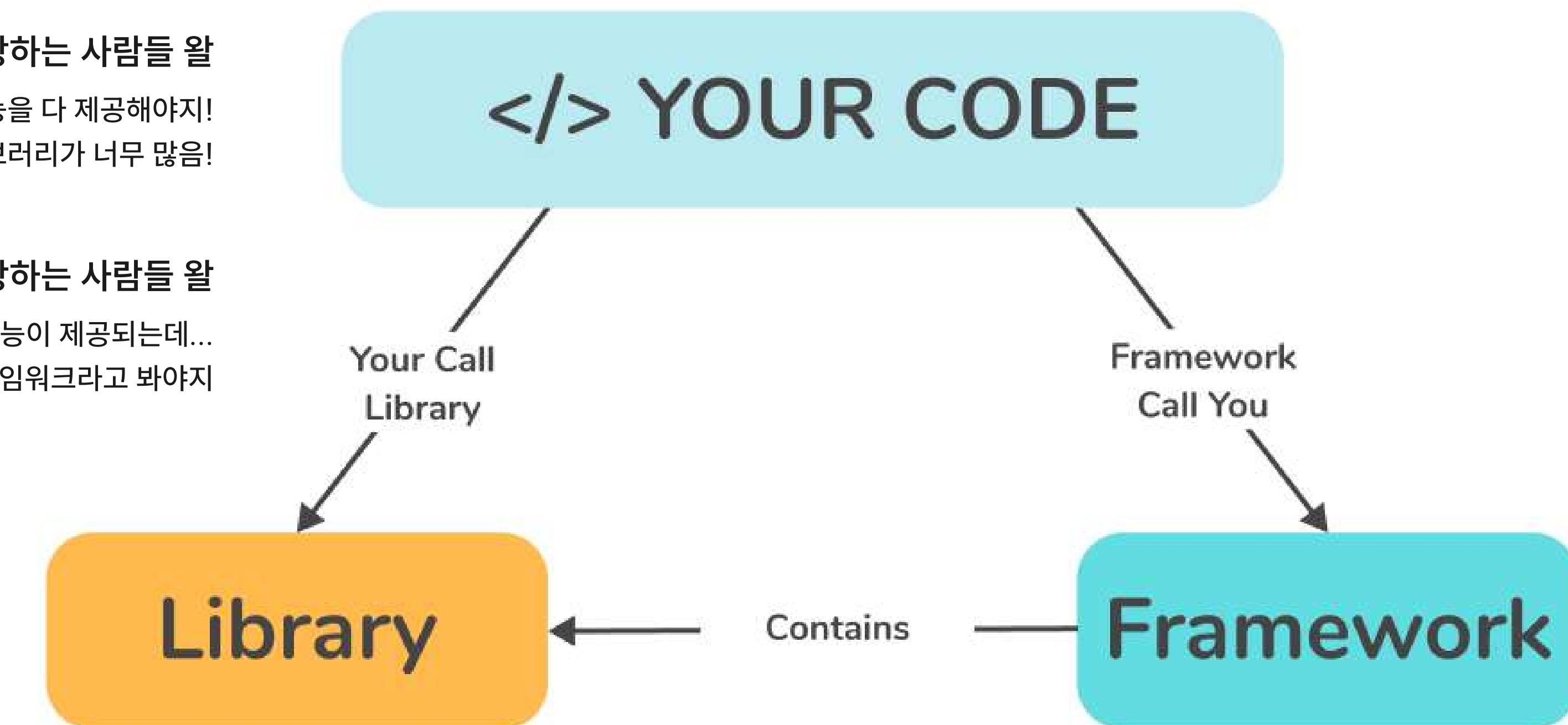
무조건 그렇다는게 아니라
본인에게 지금 필요한 방식이 무엇인가를 찾으세요 라고 하고 싶지만
특히나 렌더링과 관련된 부분은 프론트엔드 생태계에서 가장!!!!!!! 빠르게 변화하고 발전하는 부분이라(feat. 스타일링)...
지금은 과도기라고 생각합니다.

프레임워크 vs 라이브러리?

저는 개인적으로 리액트는 라이브러리라고 생각해요...(소심)

라이브러리라고 주장하는 사람들 왈
아니 프레임워크라면 기능을 다 제공해야지!
리액트는 사용하기 위해서 필수적으로 설치해야하는 라이브러리가 너무 많음!

프레임워크라고 주장하는 사람들 왈
아니 리액트랑 바닐라 자바스크립트 비교하면 얼마나 많은 기능이 제공되는데...
이정도면 프레임워크라고 봐야지



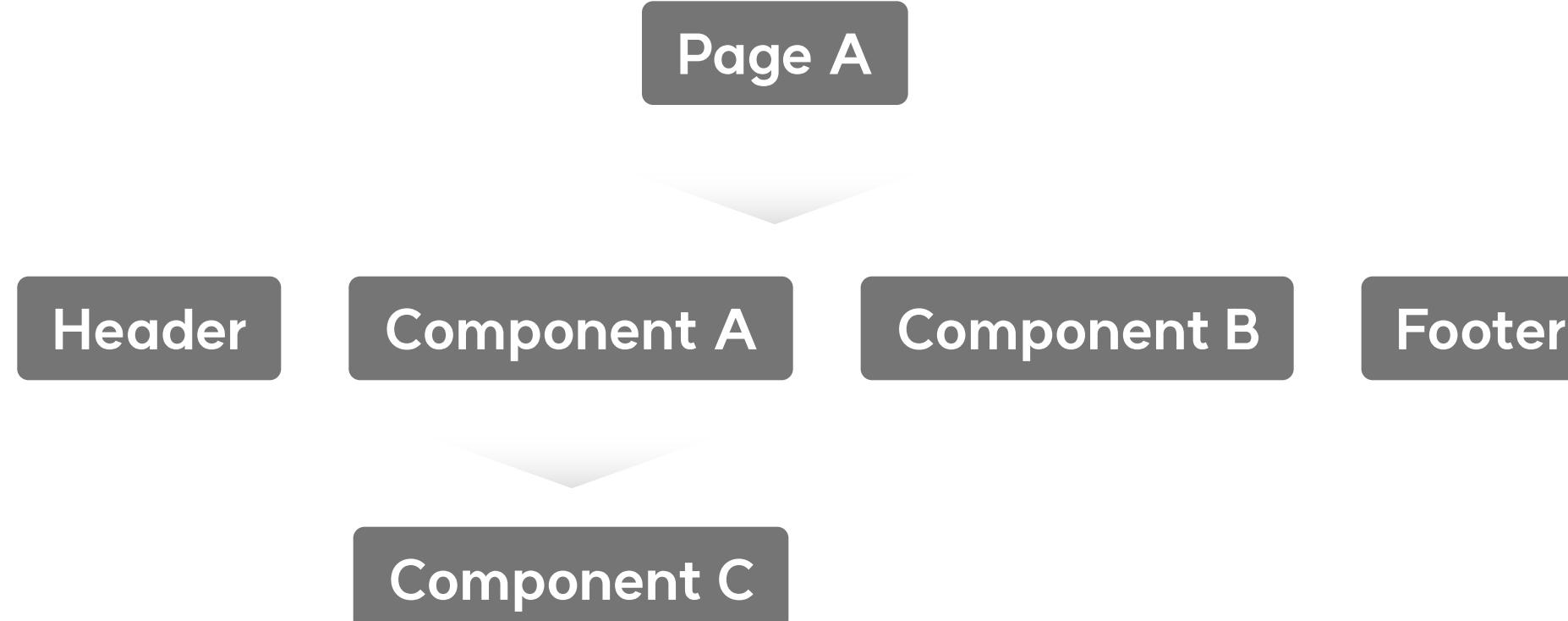
리액트의 첫 번째 특징

단방향 데이터 바인딩

데이터의 흐름이 상위 > 하위로 흐름.

역으로 거슬러 올라갈 수 없음.

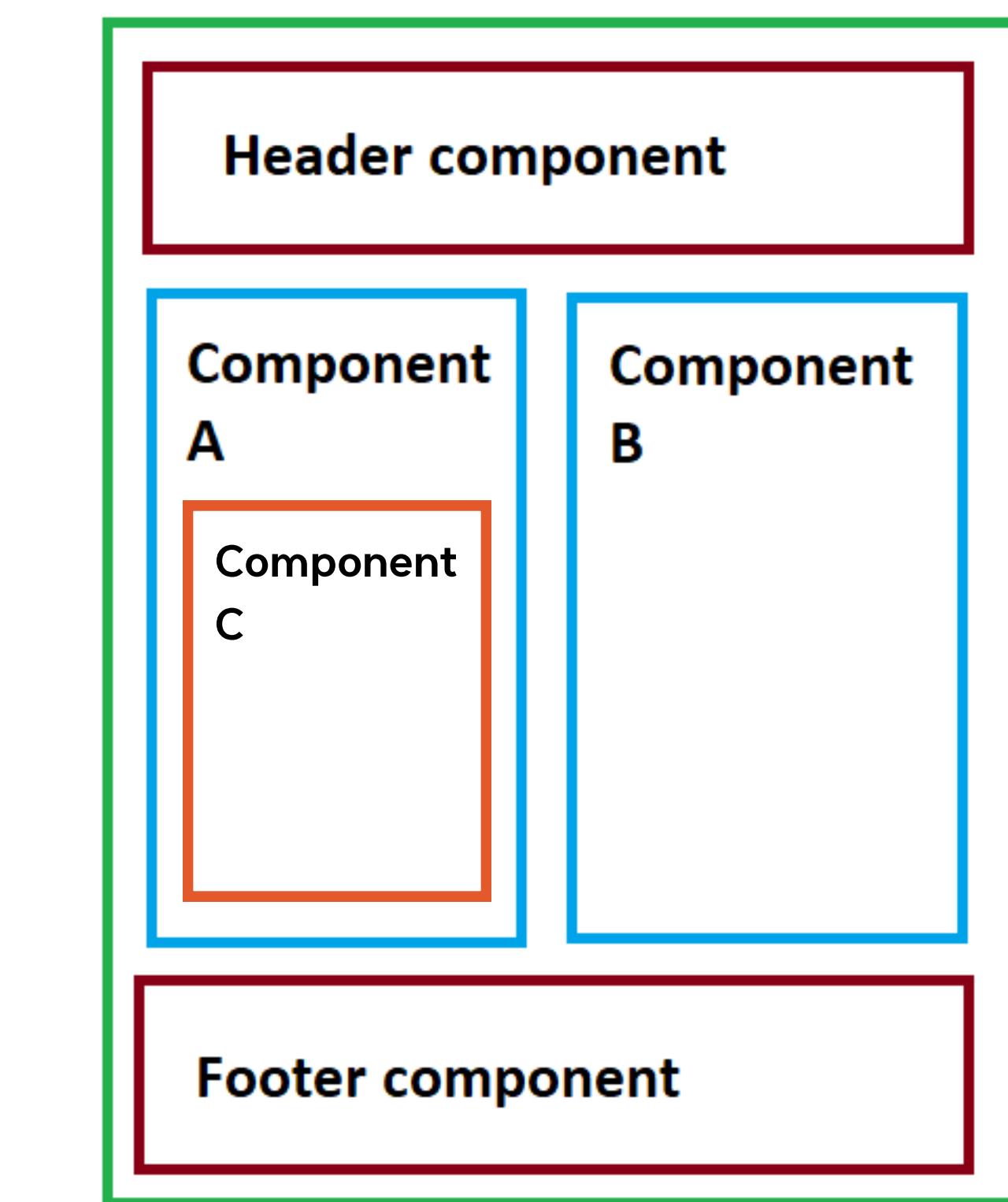
React는 컴포넌트를 기반으로 개발함



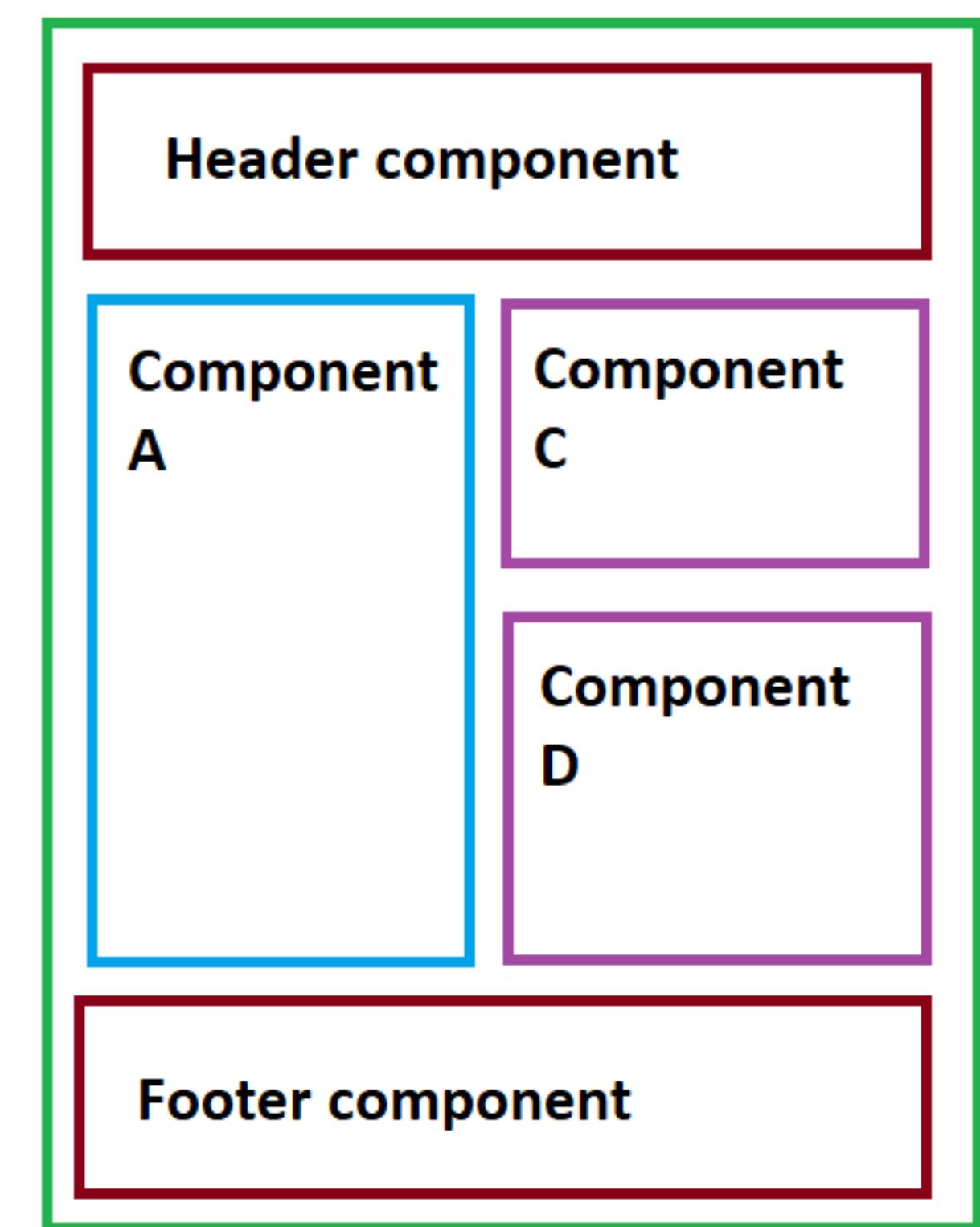
데이터 흐름이 역으로 올라갈 수 없음!
가령, Component C에서 변수를 Page A에서 수정 불가능

만약 단방향이 아니었다면...? (생각해볼까요?)

Page A



Page B

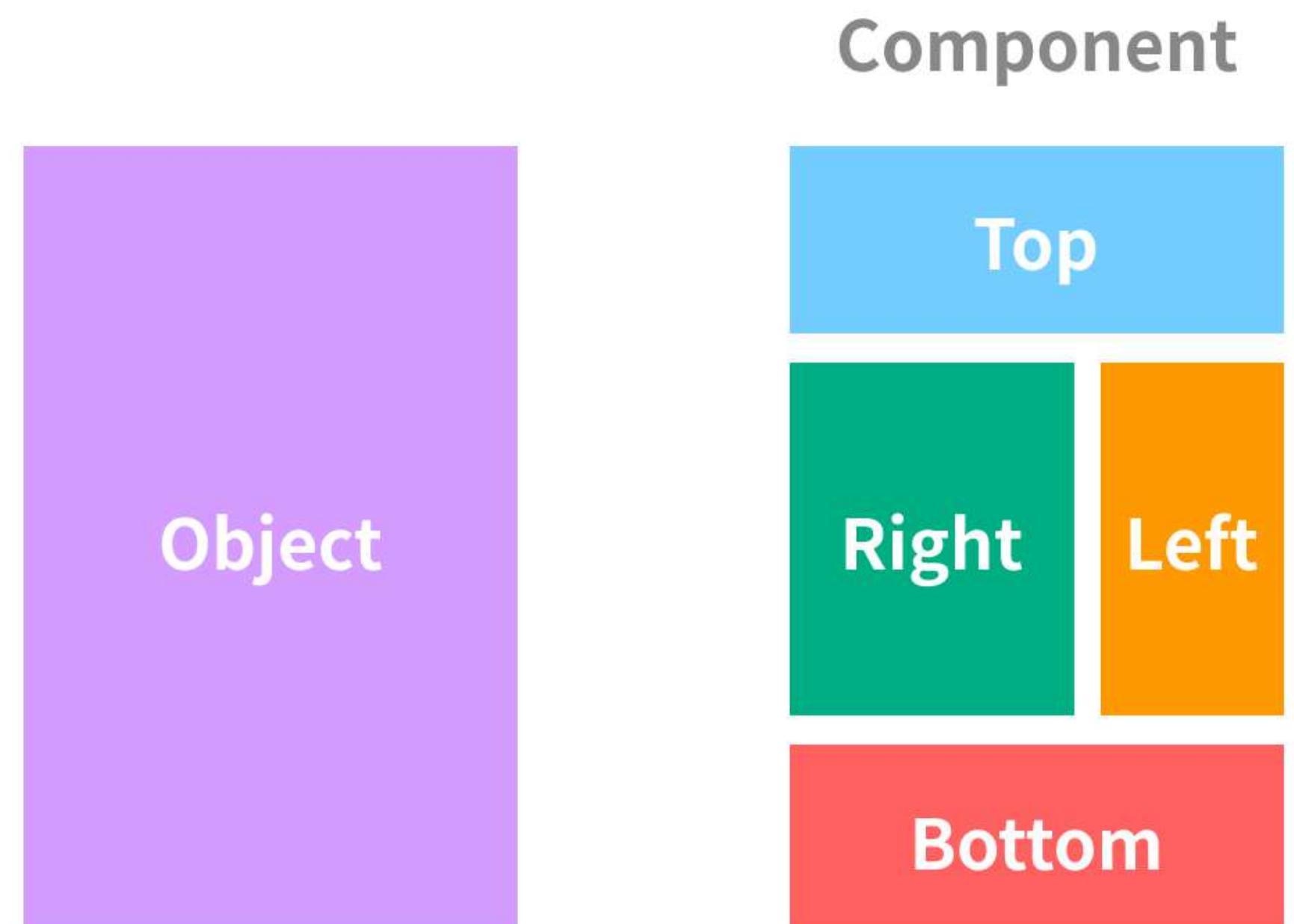


리액트의 두 번째 특징

컴포넌트 기반 구조

컴포넌트는 독립적인 단위의 소프트웨어 모듈을 뜻함.

즉, 하나의 소프트웨어에서 UI(View)를 여러개로 쪼개서 독립적인 블럭을 구분하고 이를 조립해서 화면을 구성함.



리액트의 세 번째 특징

Virtual DOM

리플로우 및 리페인트가 자주 실행되는 문제를 해결하기 위해서 화면에 표시되는 DOM과 동일한 DOM을 메모리상에 만들어놓고,
DOM 조작이 발생하면 메모리상 **가상 DOM에서 모든 연산을 수행한 후에!!!** 실제 DOM을 갱신해서 리플로우 리페인팅 연산을 최소화함

DOM (Document Object Model)

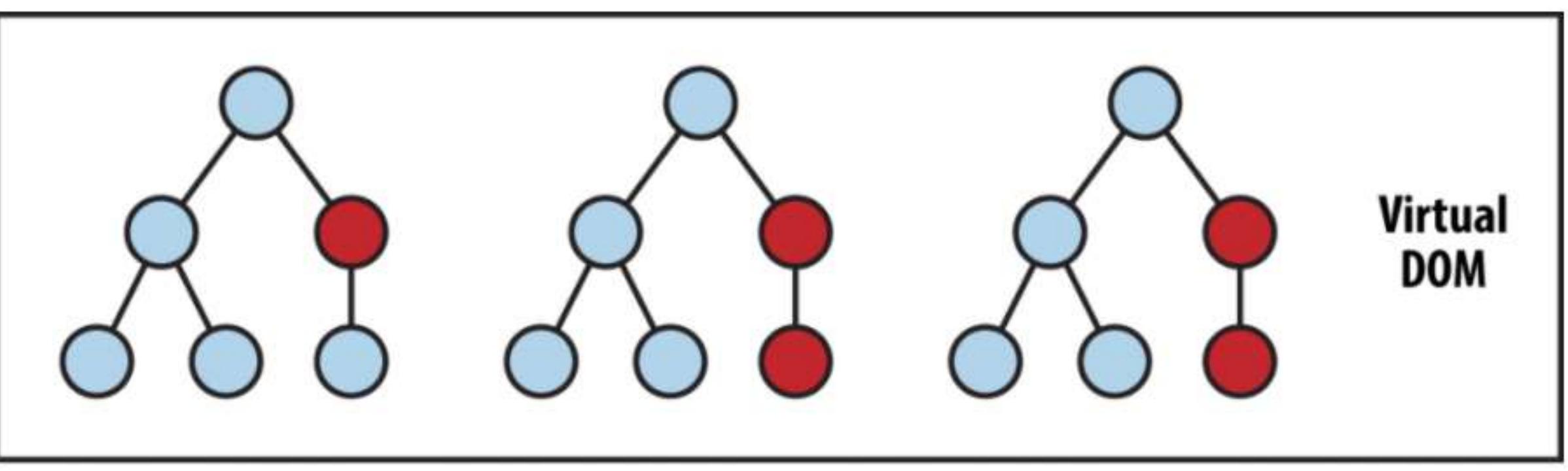
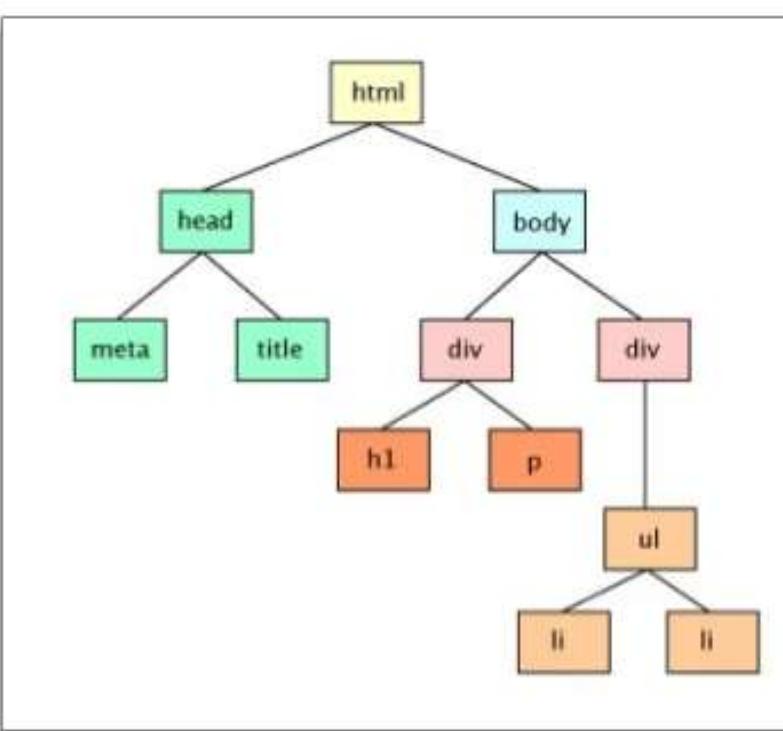
웹 브라우저가 html 페이지를 인식하는 모델의 집합

리플로우(Reflow)

DOM 조작으로 레이아웃이 다시 그려지는 과정

리페인팅(Repainting)

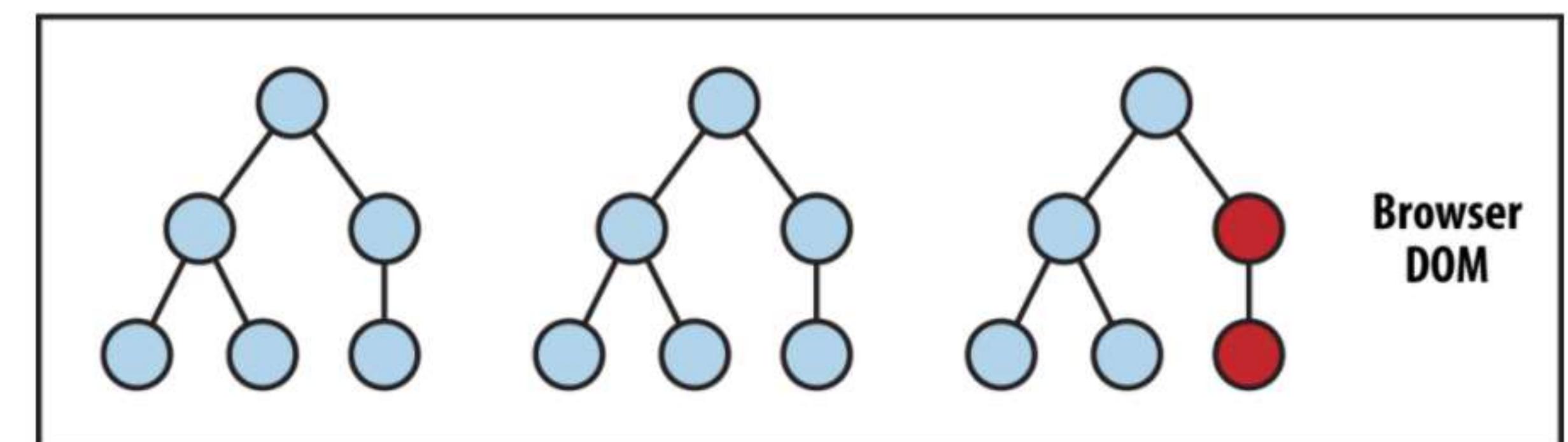
DOM 조작으로 각 DOM에 색상이 다시 입혀지는 과정



State Change → Compute Diff → Re-render

가상돔이 있기 때문에
실제로 우리가 맞이하는 브라우저 환경에서는
렌더링 성능이 굉장히 개선됨

파트장의 개인적인 의견 + 교수님들의 의견)
아직까지는 V-DOM이 최고다!
그나마 이 부분은 생태계 논란이 적은편



리액트의 네 번째 특징

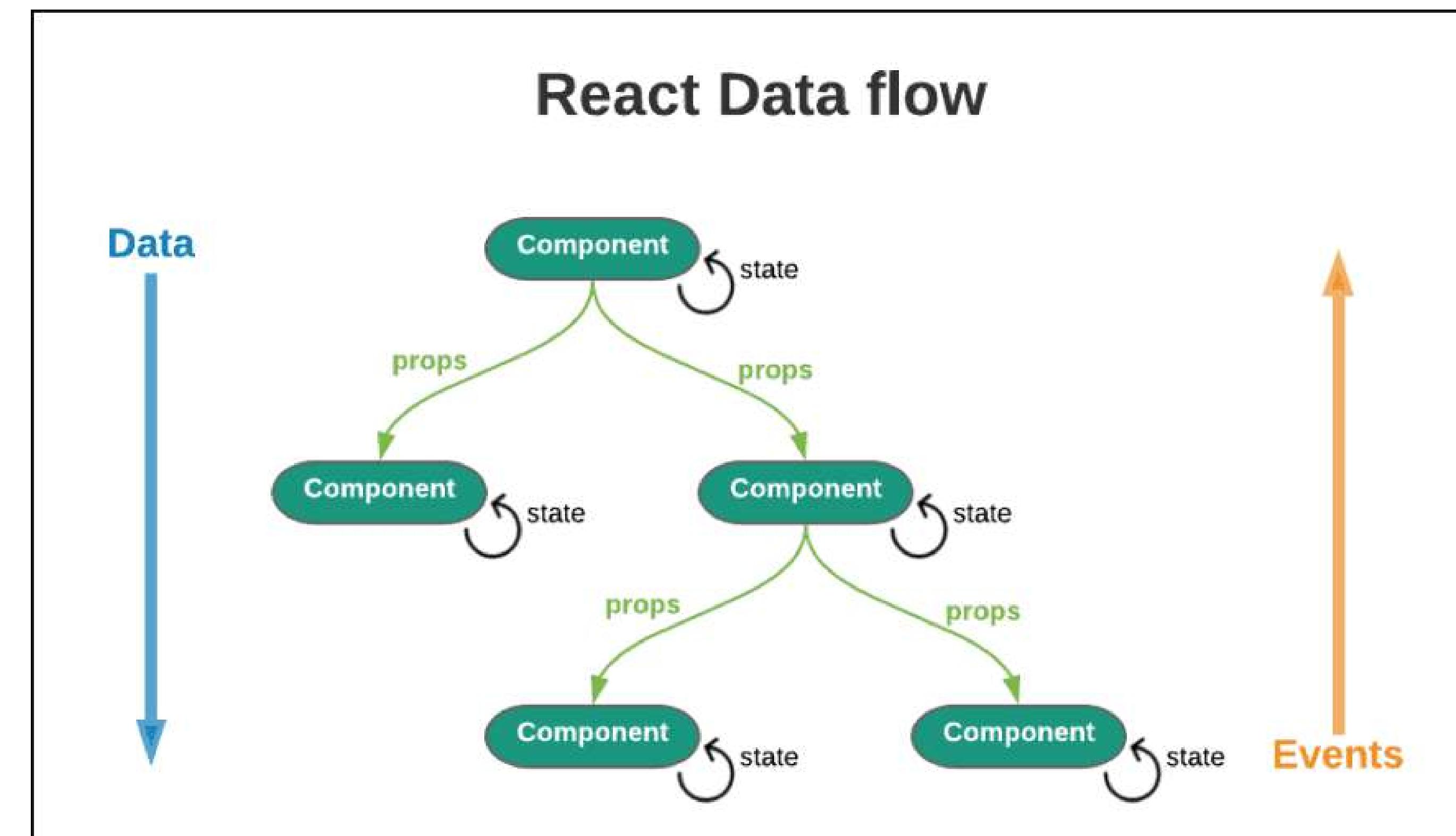
Props & State

Props) 부모(상위)에서 자식(하위) 컴포넌트로 전달해주는 아이템. 읽기 전용!

State) 컴포넌트 내부에서 선언하는 변수. 동적인 데이터를 다룰 때 사용하며, 앞으로 우리는 사용자한테 보여주는 변수를 말할 땐 State(상태)라고 할거임!

쉽게 말해서,
State는 UI에서 사용하는 변수.
변경시 DOM이 다시 그려짐

Props는 단방향 데이터 바인딩과 연관이 있음.
만약 상위 컴포넌트에서 생성한 State를
하위 컴포넌트에서 변경하고 싶다?
이때 사용하는 것이 바로 props!



부록

Props Drilling

컴포넌트 계층 구조가 복잡해지고, Depth가 깊어질수록 우리가 관리하는 State 역시 복잡해짐.
이 때, 만약 최상위 컴포넌트에서 선언한 State를 그의 10번째 자식 컴포넌트에서 수정하고 싶다면...?

Component A → Component B → Component C → ... Component K

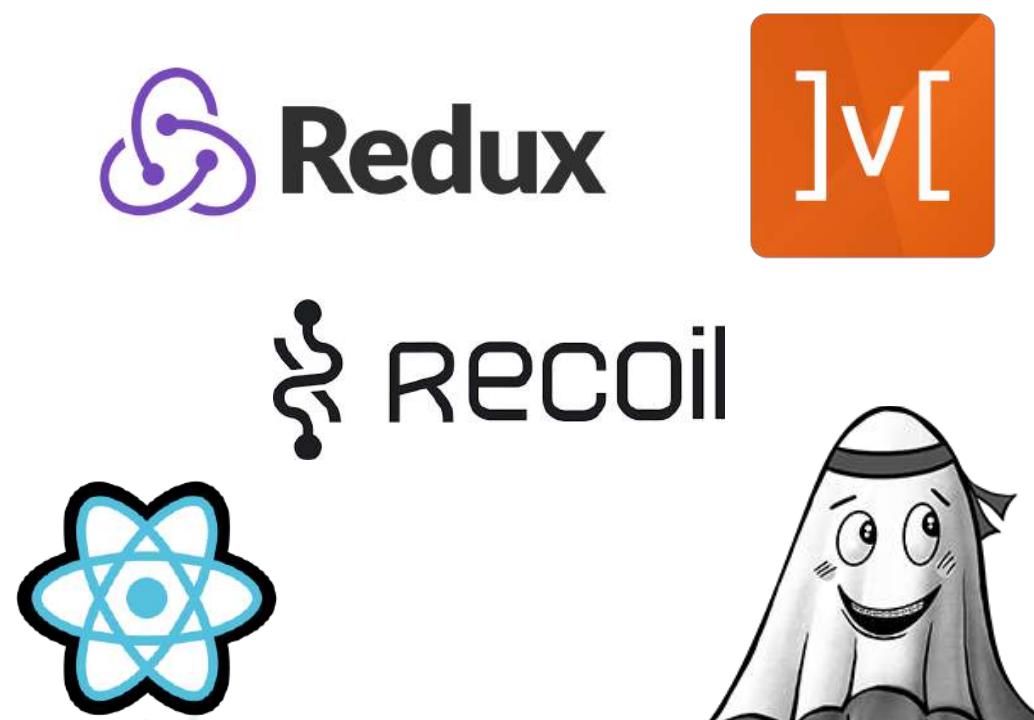
ㅁㅊ 개오바

=> 전역적으로 상태를 관리할 수 없나요?

=> 그거를 우리는 이제 "(전역)상태관리" 라고 부릅니다

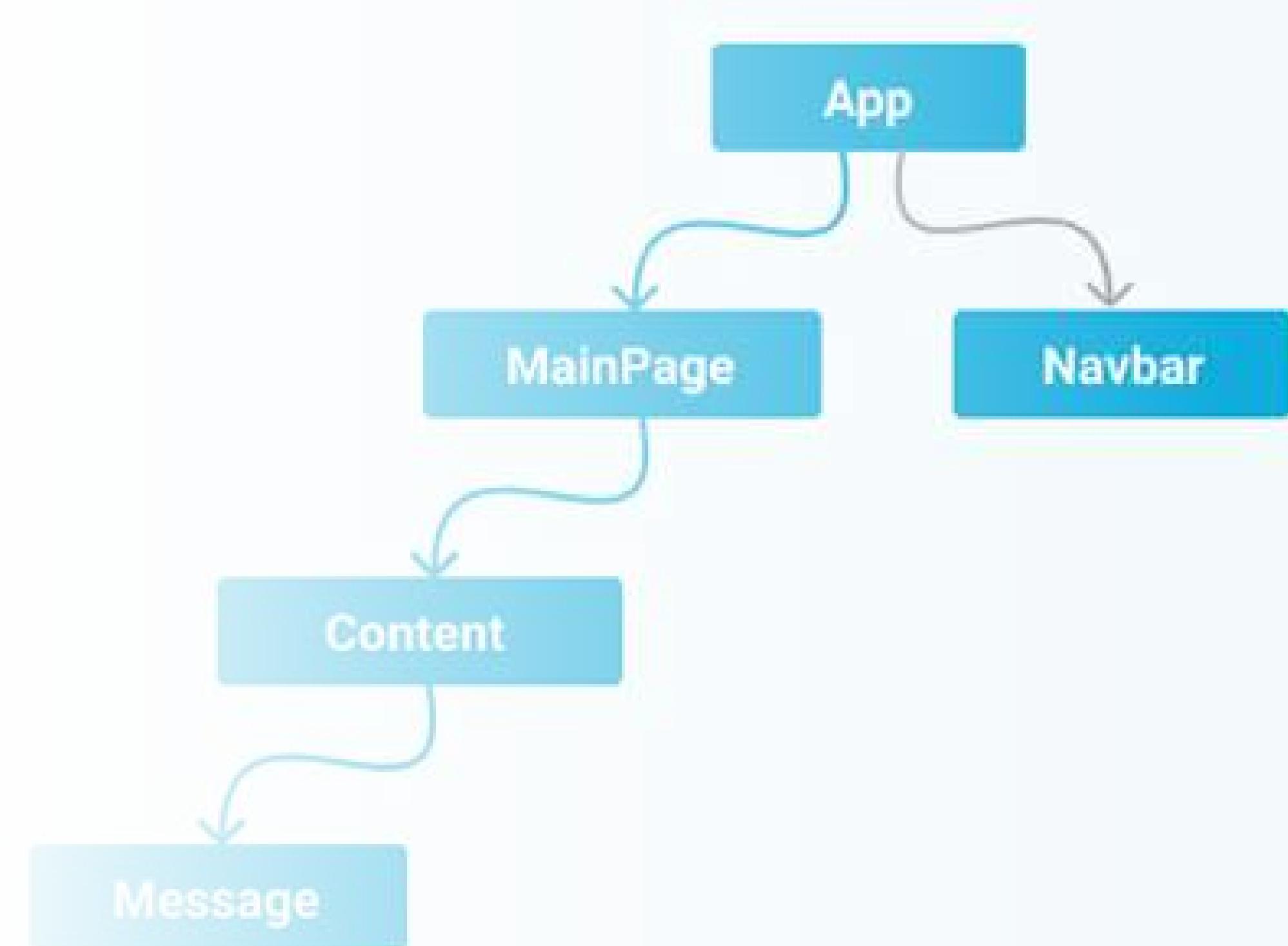
일반적으로 그냥 상태 하나하나를 관리하는 것도 상태관리라고 부릅니다.

그런데 보통 전역적으로 상태를 넓게~ 관리하는 풀로우(?)를 상태관리라고 합니다!



Context API

이런 상태관리 라이브러리들이 아주 많음
세션 끝나고 나중에 공부해보세용 (Recoil 추천)



React x javascript 파일 확장자

JSX (Javascript XML)

필수적으로 사용되어야 하는 것은 아님.

하지만 React에서 사용하는 문법이 JSX 문법!

자바스크립트 쓸 때 솔직히 HTML 자바스크립트
분리 되어있는거 이해 안되는거 인정해 안해

둘이 연결하겠다고 새로운 로직 도입하고 기술 도입하면
개빡치는거 인정해 안해

그렇다고 강제로 JSX쓰라고하면 화 잔뜩나는거
인정해 안해



[인정을 해? 안 해?
본인 화에 훈쓸려서 (인터넷을) 꺼잖아. 인정해? 안 해?]

React x javascript 파일 확장자

JSX (Javascript XML)

자바스크립트를 계승하는 중입니다...



○ㅋ 그럼 비슷하게 새로운거 하나 만들고 강제로 쓰라곤 안할게

React x javascript 파일 확장자

JSX (Javascript XML)

자바스크립트를 계승하는 중입니다...

자바스크립트 쓸 때 솔직히 HTML 자바스크립트
분리 되어있는거 이해 안되는거 인정해 안해

둘이 연결하겠나고 새로운 로직 도입하고 기술 도입하면
개빡치는거 인정해 안해

그렇다고 강제로 JSX쓰라고하면 화 잔뜩나는거
인정해 안해

이벤트 처리, State의 변화, 데이터 전처리...

자바스크립트의 로직 부분이 UI와 연결된다는거 ○ 가능합니다.
한 파일로 합칠게요

별도의 파일에 마크업과 로직 넣어서 기술을 인위적으로 분리하면..

빡치는거 ○ 가능합니다.

어차피 리액트 컴포넌트 단위로 개발하니까 컴포넌트 개발하기 쉬운 확장 문법 하나 만들게요.

아 물론 안써도 됩니다ㅎㅎ
(근데 쓰는게 편함)

JSX 문법

JSX (Javascript XML)

DOM을 구성하는 컴포넌트 파일의 경우, 반드시 하나의 Wrapper가 감싸는 형태여야함!

```
const StrategyBox = () => {
  const [curStrategy, setCurStrategy] = useState<string>(STRATIGIES[0]);
  const [curUnit, setCurUnit] = useState<string>(UNITS[0]);

  const onChangeStrategy = (str: string) => {
    setCurStrategy(str);
  };

  const onChangeUnit = (unit: string) => {
    setCurUnit(unit);
  };

  return (
    <Frame css={frameStyle}>
      <TitleWrapper>
        <StrategyDropdown
          curStrategy={curStrategy}
          setCurStrategy={onChangeStrategy}
        />
        <UnitSelector curUnit={curUnit} setCurUnit={onChangeUnit} />
      </TitleWrapper>
      {/* <StrategyChart /> */}
    </Frame>
  );
};
```

하나의 컴포넌트

return 하는 부분을 보면 DOM이니까 UI를 구성하는 컴포넌트임을 알 수 있음.
(DOM이 아니라면, util 함수)

하나의 Wrapper로 감싼 형태

return 뒤에 여러개의 DOM이 나열된 형태이면 안됨
무!조!건! 하나의 Wrapper로 감싸야함

JSX 문법

JSX (Javascript XML)

JSX 안에서 자바스크립트 표현식을 사용하려면, {} 중괄호 이용하기!

... 함수 선언하고 return 시작한 부분 생략

```
</Menu>
  {isOpen ? (
    <DropdownWrapper ref={ref}>
      {STRATEGIES.map((strategy, idx) =>
        strategy !== curStrategy ? (
          <li
            onClick={() => onChangeStrategy(strategy)}
            key={`${strategy}-${idx}`}
            css={css}
            width: 100%;
            height: 100%;
          >
            <Menu>{strategy}</Menu>
          </li>
        ) : (
          <div>
        )
      )
    </DropdownWrapper>
  ) : (
    <></>
  )
}
```

어라라?
HTML처럼 생긴 태그(DOM) 사이에서 배열의 map 메서드를..?!

으잉?
DOM 사이에 들어간 문자열인가? 근데 왜 색이 있지

중괄호를 이용해서 자바스크립트 문법을 사용하는 것.
첫 번째 경우, 배열의 map을 통해서 여러개의 DOM을 나열함
두 번째 경우, 문자열이 들어올 자리에 strategy라는 변수를 넣은 것임.

JSX 문법

JSX (Javascript XML)

지금까지 class는 잊고 className을 쓰시다!

얘네뭐임?

기존에 HTML에서 사용하던 class와 동일함
스타일링할 때 사용하는 클래스 이름임

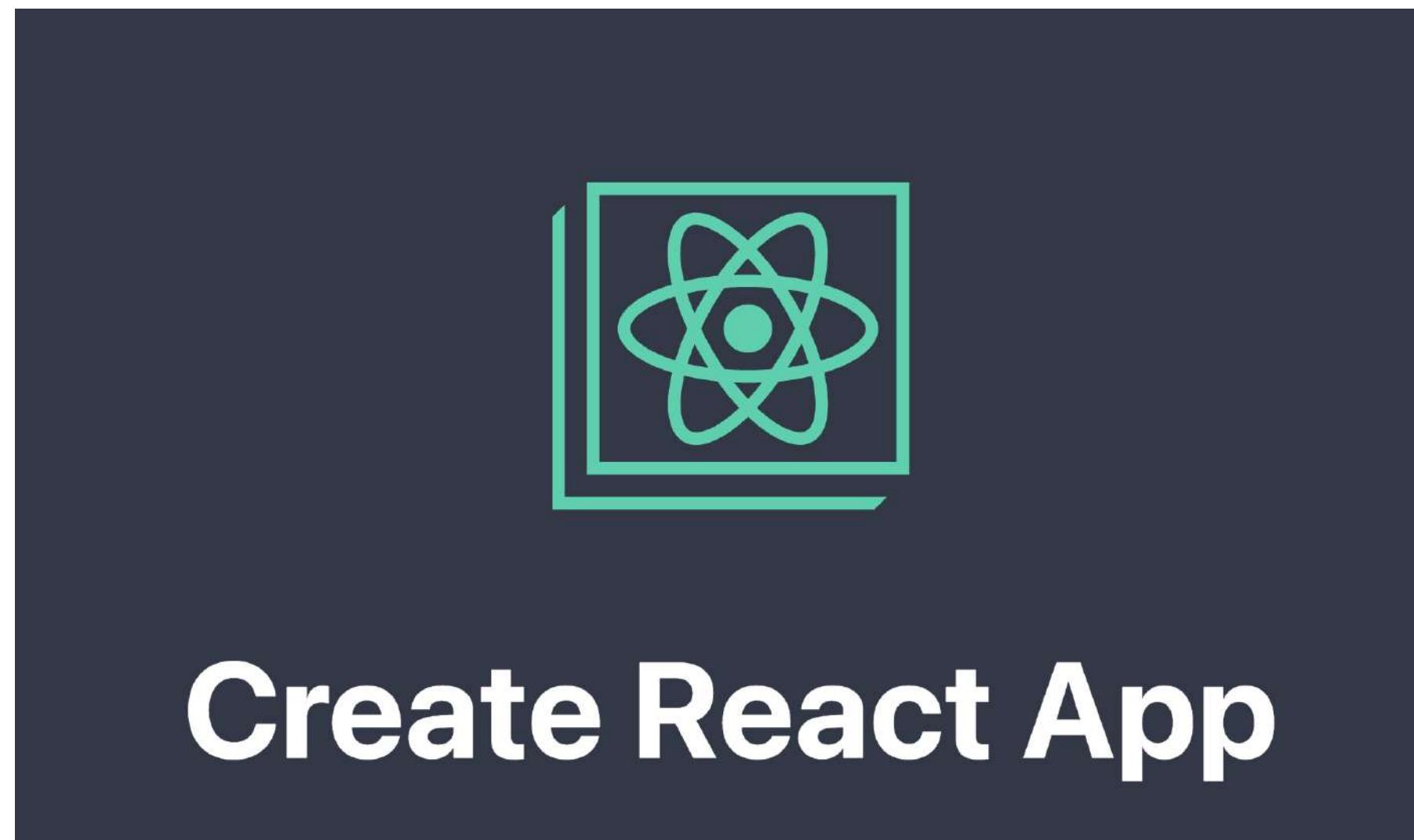
```
const EmblaCarousel: React.FC<PropType> = (props) => {
  const { slides, options } = props;
  const [emblaRef] = useEmblaCarousel(options);

  return (
    <div className="embla">
      <div className="embla_viewport" ref={emblaRef}>
        <div className="embla_container">
          {slides.map((slide, index) => (
            <div className="embla_slide" key={index}>
              <img className="embla_slide_img" src={slide.img} />
              {slide.url !== "" ? (
                <span
                  className="embla_slide_link"
                  onClick={() => window.open(` ${slide.url}`)}
                >
                  자세히보기
                </span>
              ) : (
                <></>
              )}
            </div>
          ))}
        </div>
      </div>
    </div>
  );
};
```

근데 사실 리액트 무서운 놈임

CRA (Create React App)

React라는 라이브러리는 프레임워크만큼 복잡하기 때문에,
이것을 사용하기 위해 미리 설정해주어야하는 것이 아주아주 복잡하고 어려움.
그래서 이 React를 쉽고 편안하게 바로 사용할 수 있는 Boiler-Plate를 제공하는데, 이것이 바로 CRA!



장점

자칫하면 시간이 오래 걸릴 수 있는 React 개발환경 세팅을 편하게 다 해줌!

솔직히 요즘 너무 좋아서 안쓰면 흑우임

단점

React의 구성이나 작동 원리를 정확히 모르고 쓰게됨. 아마 여러분도 그럴 것이지만 일단은 흥미를 붙이는게 우선이니까..!

나중에 뜯어보면서 공부해보셈

필요 없는 설정까지 같이 설치됨



```
$ yarn create react-app [프로젝트 이름]
```

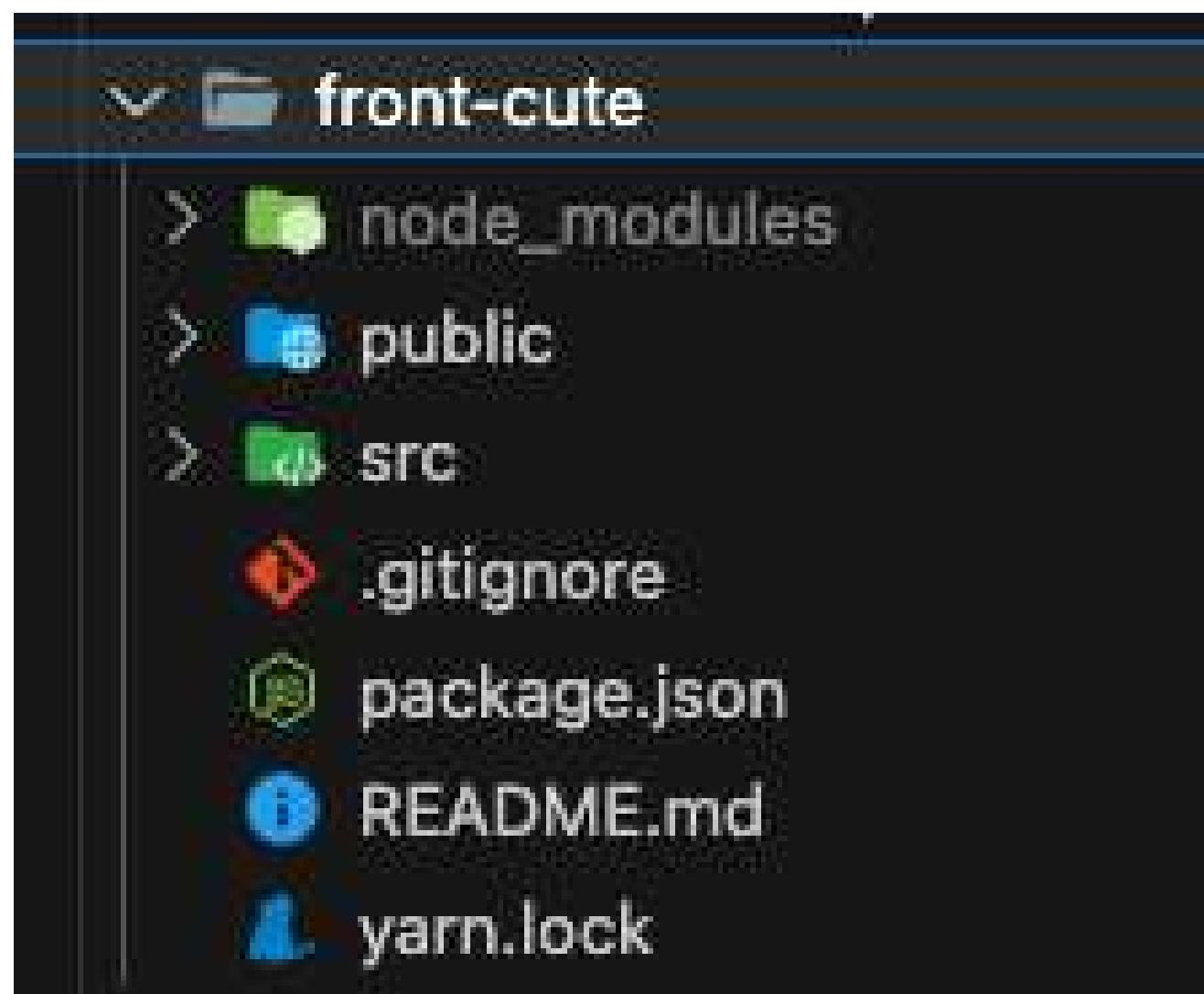
e.g. `yarn create react-app front-cute`

node_modules

해당 프로젝트에서 사용하는 패키지들이 모인 장소.

폴더 색이 회색인 이유는 git ignore 처리가 기본적으로 되어있기 때문.

왜냐하면, 하나의 프로젝트에는 정말 많은 패키지들이 있고 이 패키지들이 깃에 전부 올라가면 용량이 어마어마해짐 때문에 우리는 ‘이 프로젝트에서 이러한 패키지를 사용중이에요~’라고 알려주는 문서를 하나 만들고, 패키지 자체는 올리지 않을 것임. 어쨌든 프로젝트의 기반이 되는 패키지들이 모여있는 장소



.gitignore

git에 올리기 싫은 애들을 명시한 파일

package.json

이 프로젝트에서 이러한 패키지를 사용중이에요!라고 알려주는 파일.

우리가 원격으로 배포하는 경우 / 다른 개발자가 깃을 통해 clone하는 경우, npm이나 yarn이 이 문서를 보고 패키지를 자동으로 설치해줌

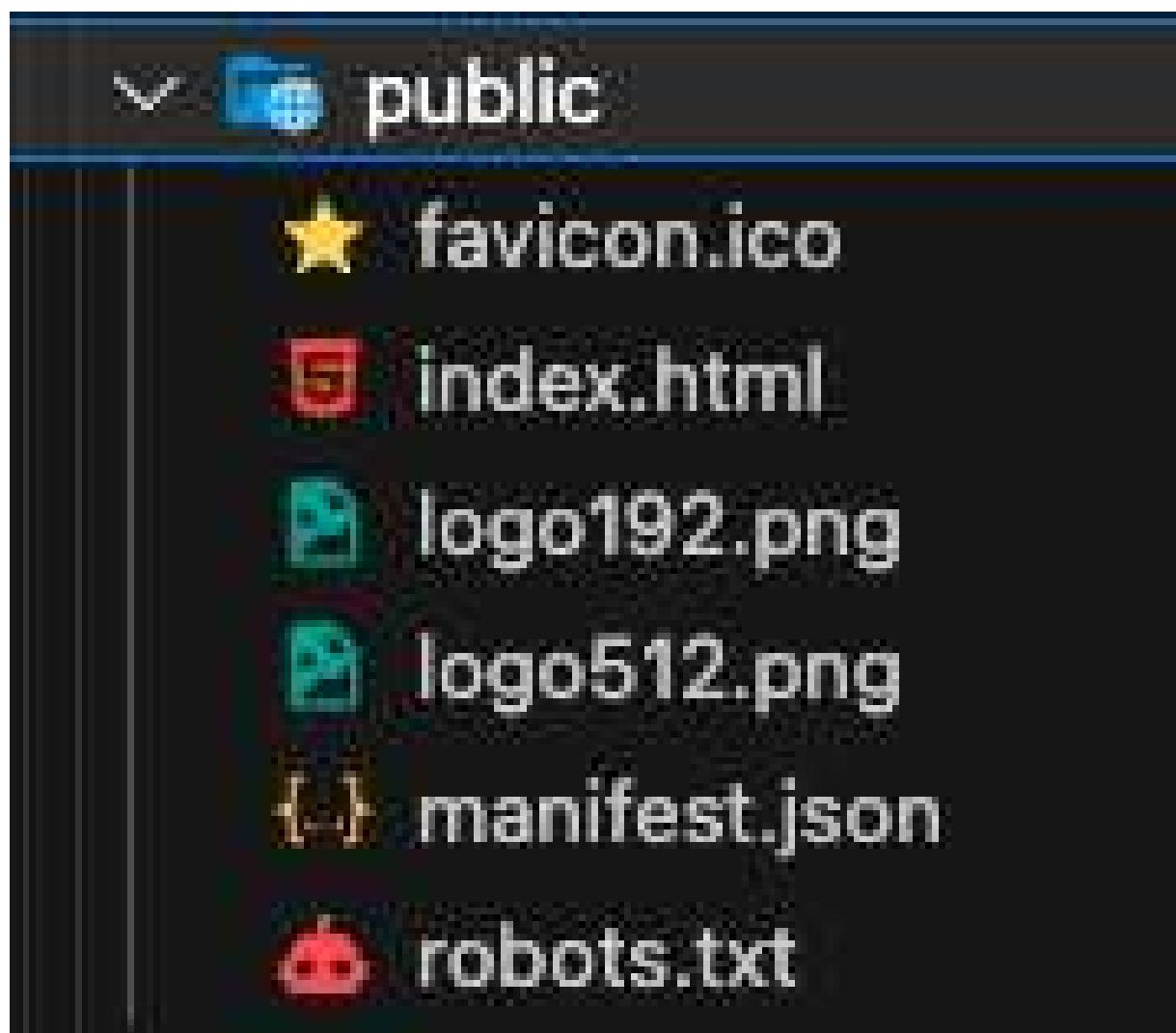
yarn.lock

npm으로 CRA하면 package-lock.json이 생길 것임. 하지만 제가 말했듯 둘이 혼용하지 말고 yarn 쓰세요 제!발!
파일이 생성된 시점의 의존성 트리에 대한 정보를 갖고 있음. 웬만하면 직접 건들 일 없음.

public

React의 정적 파일이 모여있는 장소

정적 파일이란, 이미지나 CSS HTML 등 응답할 때 별도의 처리 없이 파일 자체를 보여주는 것들



favicon.ico

탭 아이콘파일. 바꾸고 싶으면 바꿀 수 있음. index.html 수정이 필요함.

index.html

브라우저에 나타나는 하나의 HTML파일.

리액트는 CSR 방식이니까, 이 HTML에 가상돔을 주입하는 방식으로 진행됨.

manifest.json

Progressive Web App 설정 파일

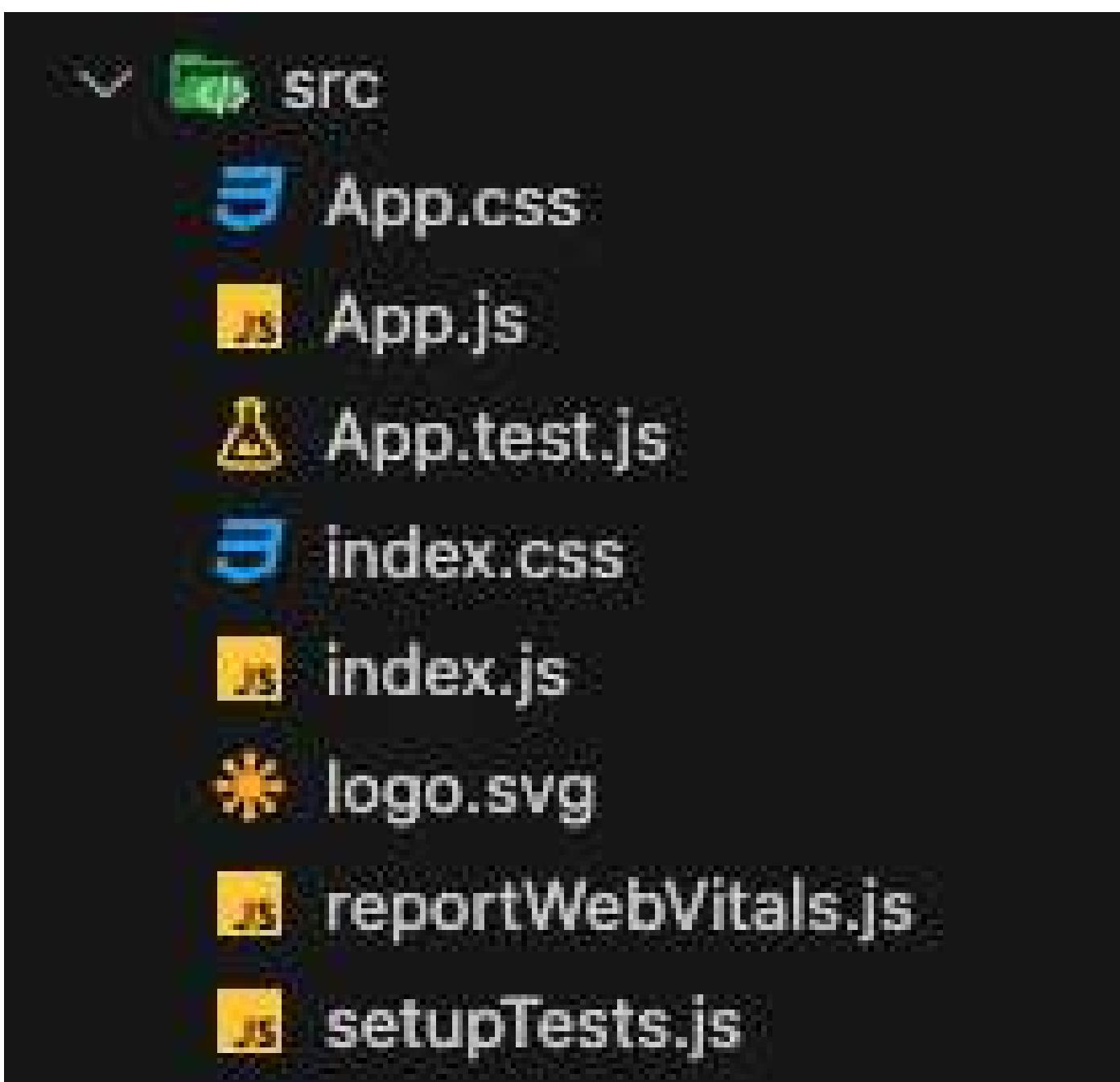
robots.txt

검색엔진 / 웹 크롤러 등 로봇의 접근을 제어하기 위한 규약 파일

src

React의 정적 파일이 모여있는 장소

정적 파일이란, 이미지나 CSS HTML 등 응답할 때 별도의 처리 없이 파일 자체를 보여주는 것들



index.js

App.js안의 App 컴포넌트와 index.html을 연결해주는 매개체

reportWebVitals.js

웹 성능을 분석해서 표현하는 객체

setupTests.js

React 프로젝트에서 테스트를 실행하기 위한 설정 파일

빌드도 하고 마

명령어

React에서 사용할 수 있는 스크립트 명령어를 확인해봅시다.

package.json에 정의 되어있음

```
▶ 디버그
{
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  }
}
```



(yarn) start

개발 서버 실행.

소켓을 이용한(핫-리로드) 로컬 개발 서버가 자동으로 실행됨. 포트는 3000번이 기본 값.

~eject는 하지마세요~

(yarn) build

프로덕션용 빌드 생성

배포용 프로덕트를 생성함. package.json 상단의 속성을 통해 버전 관리 가능.

구구절절 이야기를 하자면, CRA의 장점은 리액트의 복잡한 설정 파일들이 숨겨져있다는 것. 때문에 개발자인 우리가 그 설정파일을 하나하나 건들 필요가 없고 업데이트 할 필요도 없는건데 eject 명령은 그 설정파일을 썩다 분리해서 개발자가 관리하겠다는 뜻.

한 번 실행하면 이전으로 돌아가는 것도 불가능함.

설정을 커스텀하려면 craco와 같은 라이브러리가 있으니까... eject는 최악의 경우에만 쓰는걸로

(yarn) test

테스트 스크립트를 실행함.

~~~.test.js 파일들 안에 테스트 코드를 작성하고, 이 코드들을 실행



React에서 컴포넌트를 생성하는 두 가지 방법

## 클래스형 컴포넌트 vs 함수형 컴포넌트

과거에는 클래스형 컴포넌트와 함수형 컴포넌트에 장단점이 있었음.

이제는 함수형 컴포넌트가 클래스형 컴포넌트의 장점을 흡수해버림

요컨대 함수형 컴포넌트 쓰면 됩

```
import React, {Component} from 'react';

class App extends Component {
  render() {
    const name = 'react';
    return <div className="react">{name}</div>
  }
}

export default App;
```

메모리 자원을 클래스형 컴포넌트보다 훨씬 덜 사용함  
빌드한 결과물의 크기가 클래스형 컴포넌트보다 작음  
보기도 편함

```
const MyComponent = ({ name, children }) => {
  return (
    <div>
      안녕하세요 제 이름은 {name}입니다.<br />
      children 값은 {children}입니다.
    </div>
  );
};
```

## 본격적으로 개발 서버를 실행해봅시다!

개발 서버를 실행해보고, 쓸모 없는 파일을 정리해볼게요!

가보자고!



## 간단한 컴포넌트를 만들어봅시다!

리액트에서 컴포넌트를 어떻게 만들고, 호출하는지 배워봅시다.

제가 모시겠습니다



## props를 전달해봅시다!

state 선언은 이따 해볼게요! prop를 전달하면서 리액트의 단방향 바인딩을 느껴봅시다



## props를 전달해봅시다!

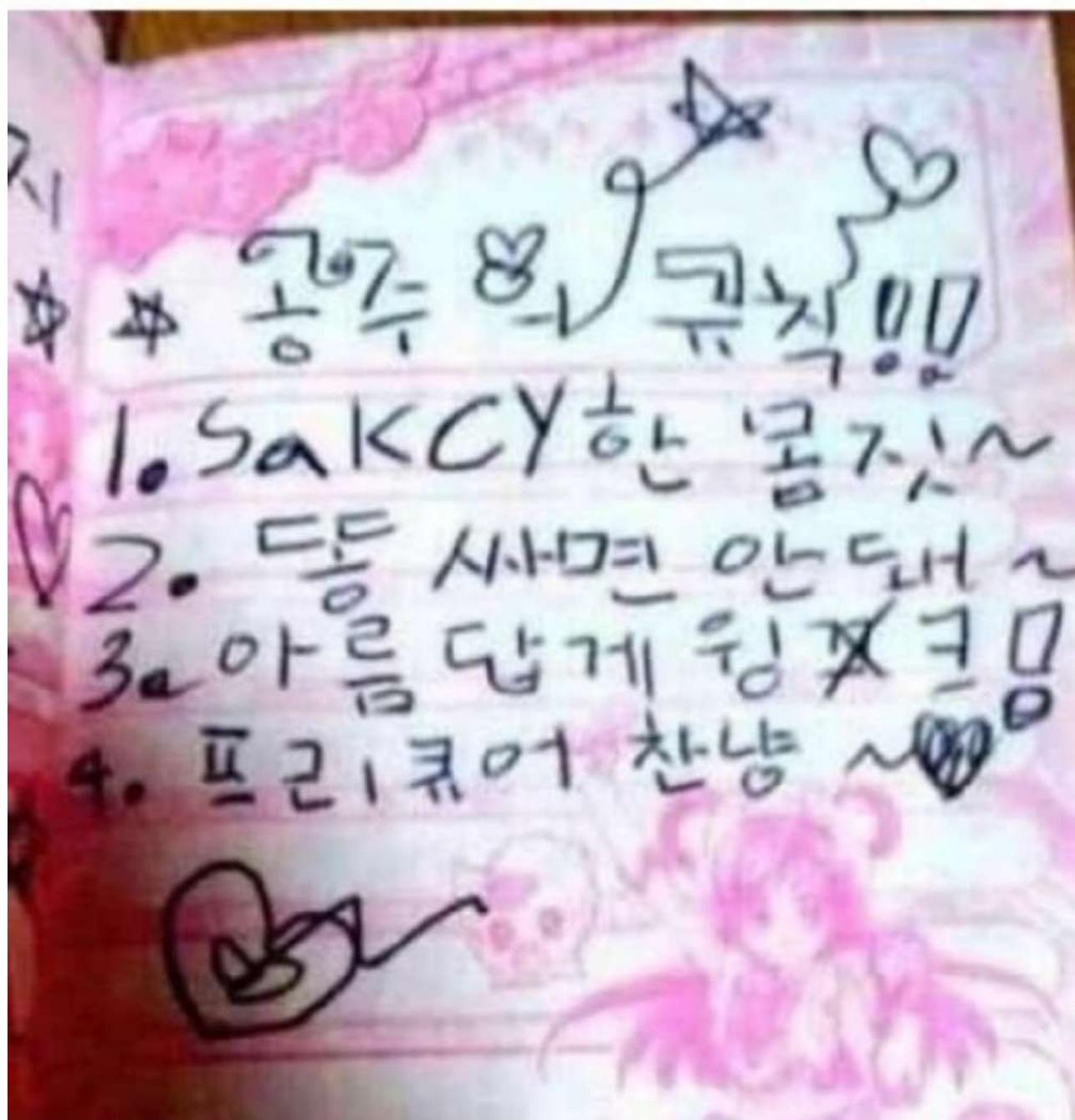
state 선언은 이따 해볼게요! prop를 전달해봅시다!



# Hooks

과거에 쓰던 클래스형 컴포넌트의 기능들을 함수형 컴포넌트에서 사용할 수 있도록 만든 모듈

React 16.8부터 이용 가능



## Hooks의 규칙~❤

### 1. 최상위 스코프에서만 Hook을 호출해야한다.

- 반복문이나 조건문 혹은 중첩 함수 내에서 Hook을 호출하면 안됨.
  - 아까 봤던 함수 컴포넌트 자체에서만 선언할 수 있음
- Hook은 호출하는 순서에 영향을 받음.
  - 조건문이나 반복문 등 논리적으로 이상이 생길 수 있는 부분을 잘 고려해야함.

### 2. 리액트 함수 내에서만 Hook을 호출해야한다.

- 일반 js 함수에서 호출 금지
- 함수형 컴포넌트나 Custom Hook을 만들 때 호출이 가능함

이것만 지키면 hook을 잘 활용할 수 있음!

그리고 웬만하면 저하고 경우에 리액트가 오류 내뿜ㅋㅋ

React Hooks

## useState()

State를 선언할 수 있는 Hook

```
const [상태 이름, 상태 Setter] = useState(초기 값);
```



e.g. `const [count, setCount] = useState(0)`

setState는 camelcase로 작성합니다

초기값은 최초 렌더링시 상태에 저장되는 값!

숫자, 문자, 객체, 논리값, 배열... 대부분의 값을 상태에 넣을 수 있음.

React Hooks

## useState()

State를 선언할 수 있는 Hook

```
const [상태 이름, 상태 Setter] = useState(초기 값);
```

→ e.g. const [count, setCount] = useState(0)

setState는 camelcase로 작성합니다

초기값은 최초 렌더링시 상태에 저장되는 값!

숫자, 문자, 객체, 논리값, 배열... 대부분의 값을 상태에 넣을 수 있음.

→ 네! 맞아요!



오잉 근데 const 쓰면 값 못바  
꾸잖아요

React Hooks

## useState()

State를 선언할 수 있는 Hook

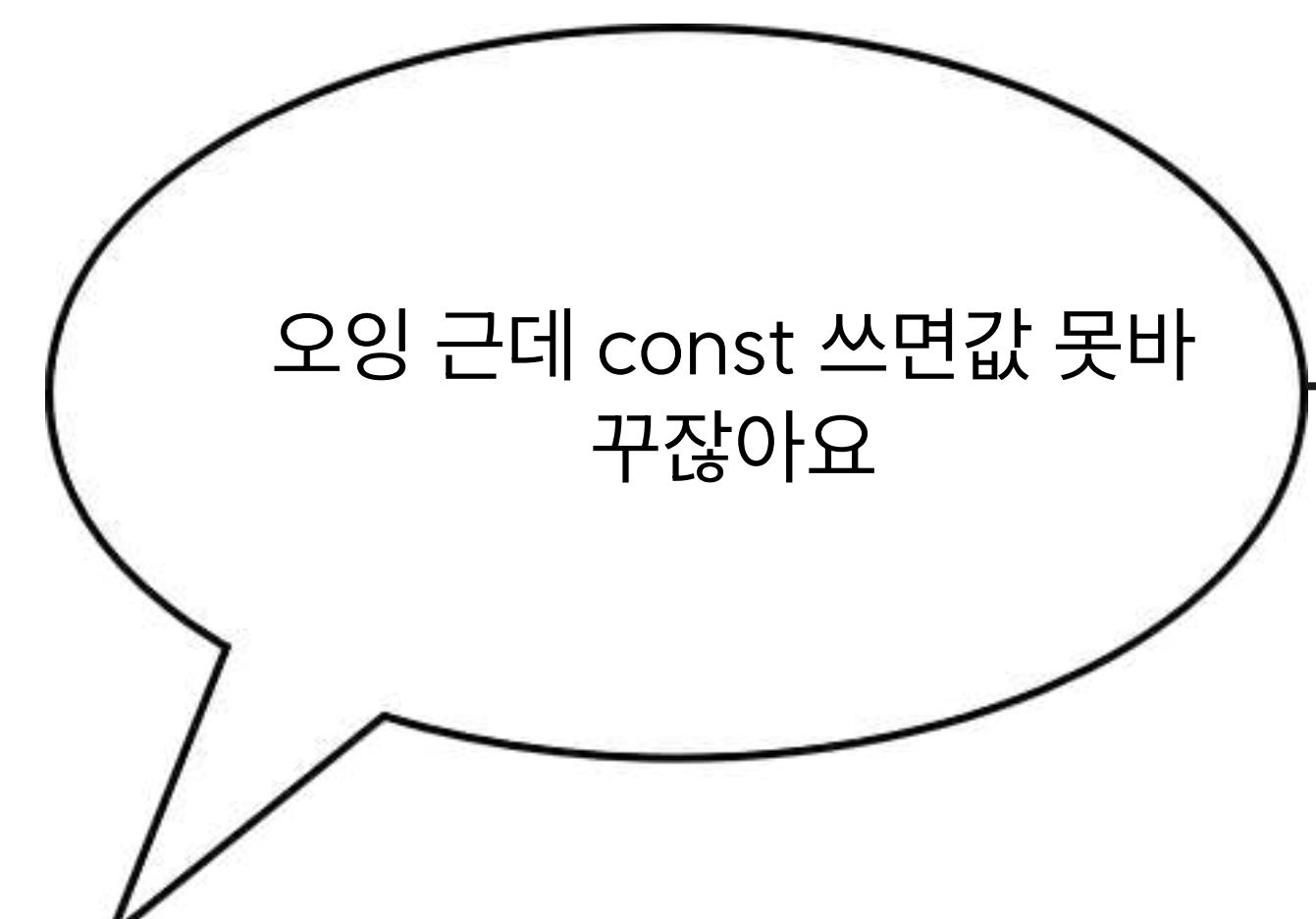
```
const [상태 이름, 상태 Setter] = useState(초기 값);
```

→ e.g. const [count, setCount] = useState(0)

setState는 camelcase로 작성합니다

초기값은 최초 렌더링시 상태에 저장되는 값!

숫자, 문자, 객체, 논리값, 배열... 대부분의 값을 상태에 넣을 수 있음.



ㅋㅋㅋㅋ

상태는 const로 선언되는게 맞아요!

대신 수정을 못하는 것도 맞습니다.

그래서 setState가 있는거예요!

값을 수정한다고 생각하지 말고,

setState라는 메서드를 이용해서 재할당한다고 생각하면 됩

```
const [count, setCount] = useState(0);
count = 1;
```

이렇게 써면 안돼유!

```
const [count, setCount] = useState(0);
// count = 1; 땡땡 완전 틀렸음
setCount(1); // 담동댕
```

이렇게 값은 바꿔줄 수 있음!

React Hooks

## useEffect()

특정 값의 변경을 감지해서 콜백 함수를 실행하는 메서드

useEffect(콜백 함수, [감지하고 싶은 값들 배열])

→ e.g. `useEffect(() => console.log(count), [count]);`

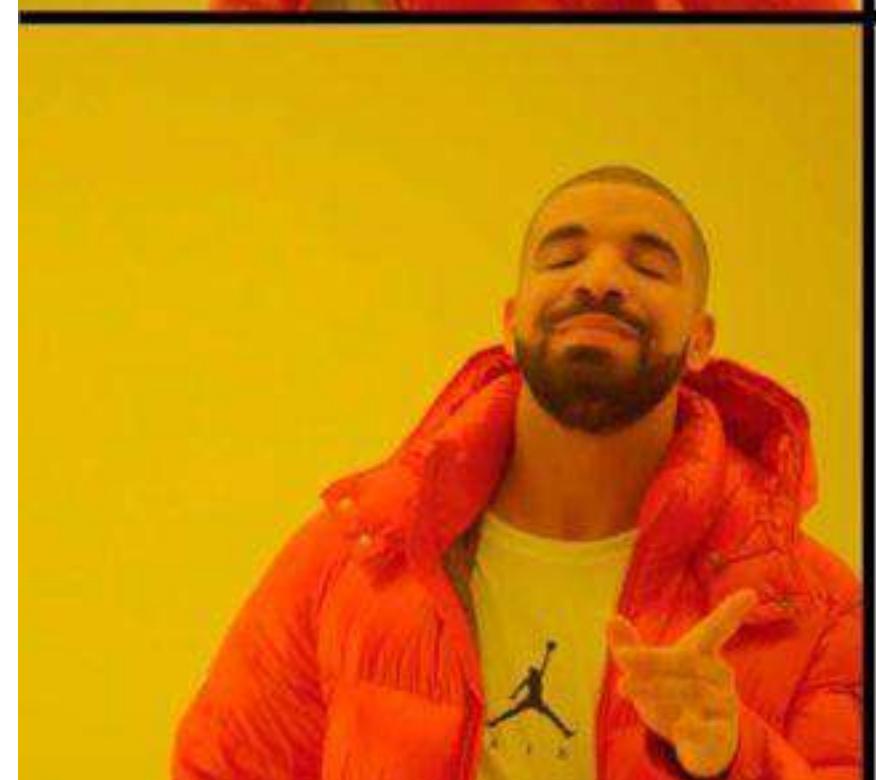
count의 값이 바뀔 때마다 count를 호출함

클래스형일 때는 라이프 사이클을 핸들링하기가 어려웠음

최초 렌더링시 무조건 한번 콜백함수 실행됨!

만약 의존성 배열에 아무것도 넣지 않는다면 최초 실행 이후 실행되지 않음.

이 useEffect의 가치는 직접 실습하면서 알아보도록 합시다!



`componentDidMount()`  
`componentDidUpdate()`

`useEffect()`

## React Hooks

## useRef()

특정 DOM에 직접적으로 접근하거나 DOM을 직접 생성하는 경우에, 특별히 사용하는 함수 (바닐라 자바스크립트에서 querySelector와 같이!)

```
const 변수명 = useRef(초기 값)
```

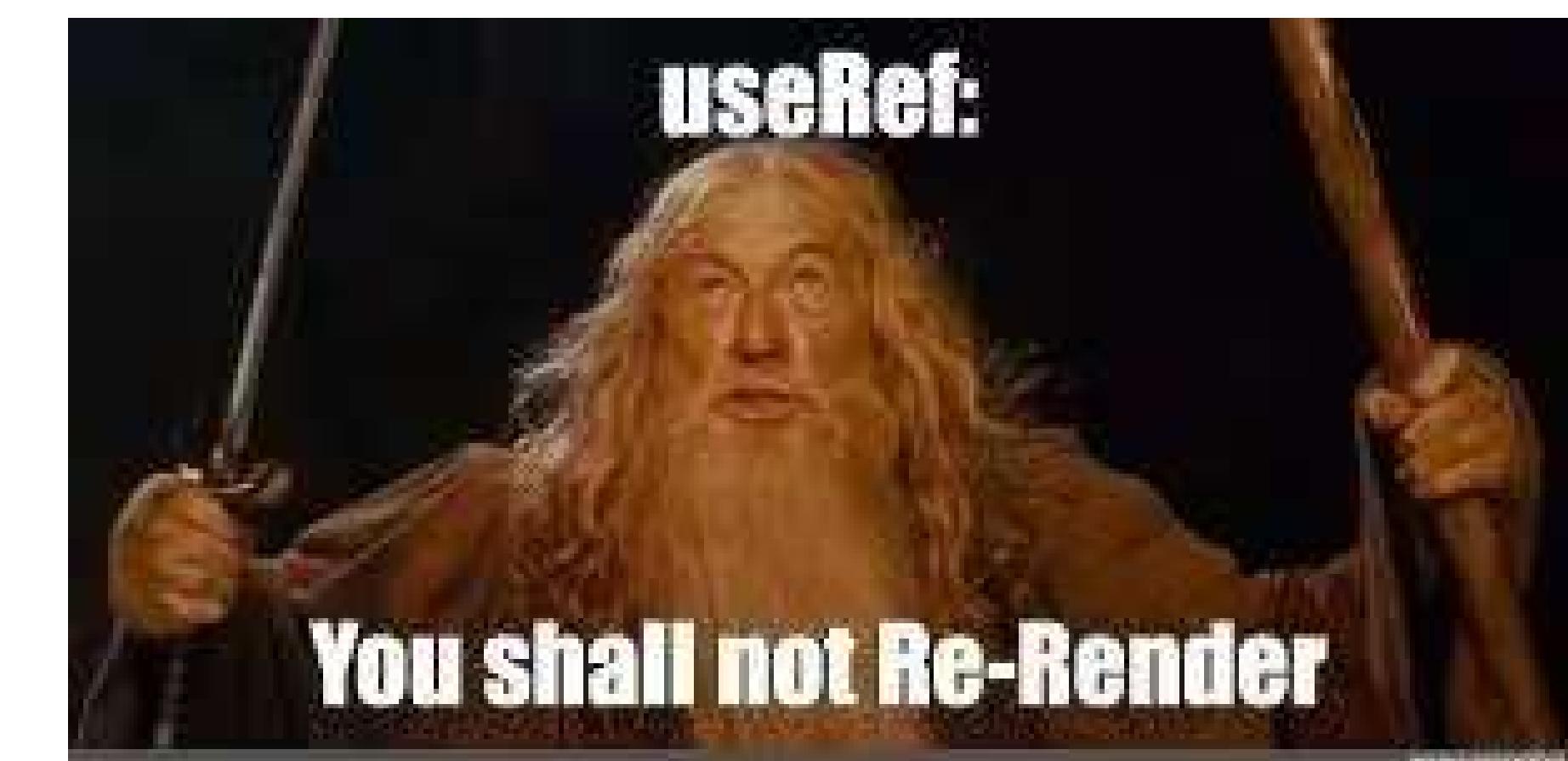
→ 변경 가능한 참조 객체를 반환함.

젠향공간 혹은 DOM 위치에 접근할 때 사용함

React에서는 가상 DOM을 쓰기 때문에 바닐라의 querySelector등을 사용할 수 없음. 때문에 참조 객체를 생성해서 수정하는 방식

특정 DOM의 참조 값을 변경하는 경우가 아니면, 값이 변경되어도 리렌더링 되지 않음

**ref.current.focus() 실습을 통해서 무슨 느낌인지 감만 잡아봅시다!**



React Hooks

## 실습 드가자

useState와 useEffect는 정말 ... 꼭 알아야해요 ...

useRef는 자주 쓰이진 않지만 DOM을 복잡하게 핸들링하거나, 성능을 개선하는 상황에서 자주 쓰입니다. 무슨 역할인지만 알고 넘어가도록해요!

- 할 일 -



React Hooks

## 사실 그녀석은 최약체였다구?

실습을 잘 마무리했지만... useState useEffect useRef는 최약체였다구요?



"처음 뵙겠어요 오라버니.  
이 탑의 4층까지 용케 올라오셨네요.  
하지만 여기까지예요.  
나, 사슬낫의 제니가 상대니까"



각성한 사슬낫의 제니마저 쓰러뜨리다니, 감탄이 절로  
나오는군요. 하지만 더이상의 장난은 없답니다. 바로 나  
녹말이쑤시개의 정려원이 있으니까요.



뭐?!!! 녹말 이쑤시개의 정려원을 해치우다니  
제법이구나!!! 그럼 나 동앗줄의 나문희가  
**상대해주지....!!!**

react@18.2.0  
useInsertionEffect

Hooks

useCallback

useContext

useDebugValue

useDeferredValue

useEffect

useId

useImperativeHandle

useInsertionEffect

useLayoutEffect

useMemo

useReducer

useRef

useState

useSyncExternalStore

useTransition

React Hooks

## 나중에 공부해보세요

useCallback | useMemo | React.memo | useContext | useReducer | 커스텀 훅

지금 하면 님들 기절함



저는 그만 정신을  
잃고 말았습니다

다시 정신 차리시고



하지만 저는 다시 정신  
을 차렸습니다

마지막 실습 드가자

## useState()와 useEffect()를 이용한 카운터 만들기

10분동안 스스로 작성해보고, 함께 만들어봅시다! (스타일링 적용 안해도됨 다음주에 스타일링만 할거임)

Current Number is...



5

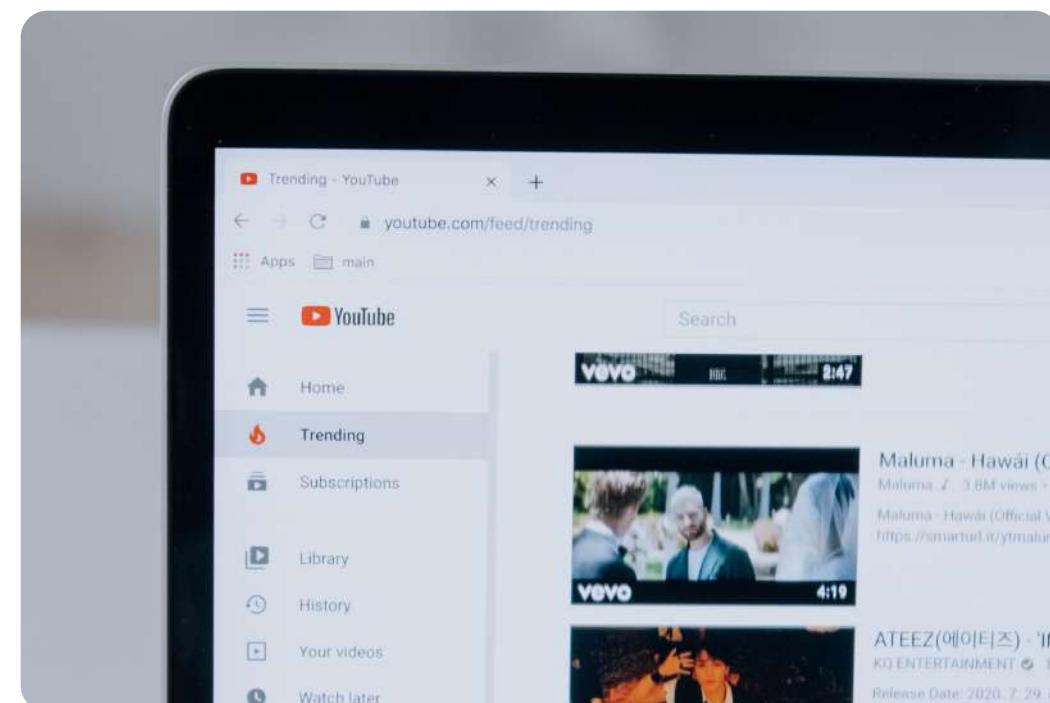


**과제공지**

## 마지막 클론코딩!

3주 동안 클론코딩(퍼블리싱)하느라 고생 많았습니다!

이번 과제까지만 클론 퍼블리싱 진행할거구 다음주에는 직접 디자인도 하고 그거 개발할거임 이제 메인디쉬ㅋㅋ



3주차 고정과제

### 유튜브 랜딩 페이지 클론

유튜브에 접속했을 때 최초로 접하는 홈 피드 페이지를 클론하세요!

영상이 아닌 이미지로 구성하세요!

하나의 페이지를 단순히 HTML과 CSS만으로 클론하면 돼요.

[유튜브 랜딩 페이지로 이동](#)

**다들 고생 많았어요** 😍

세션 휴강 조사하기 (이준규기억해)