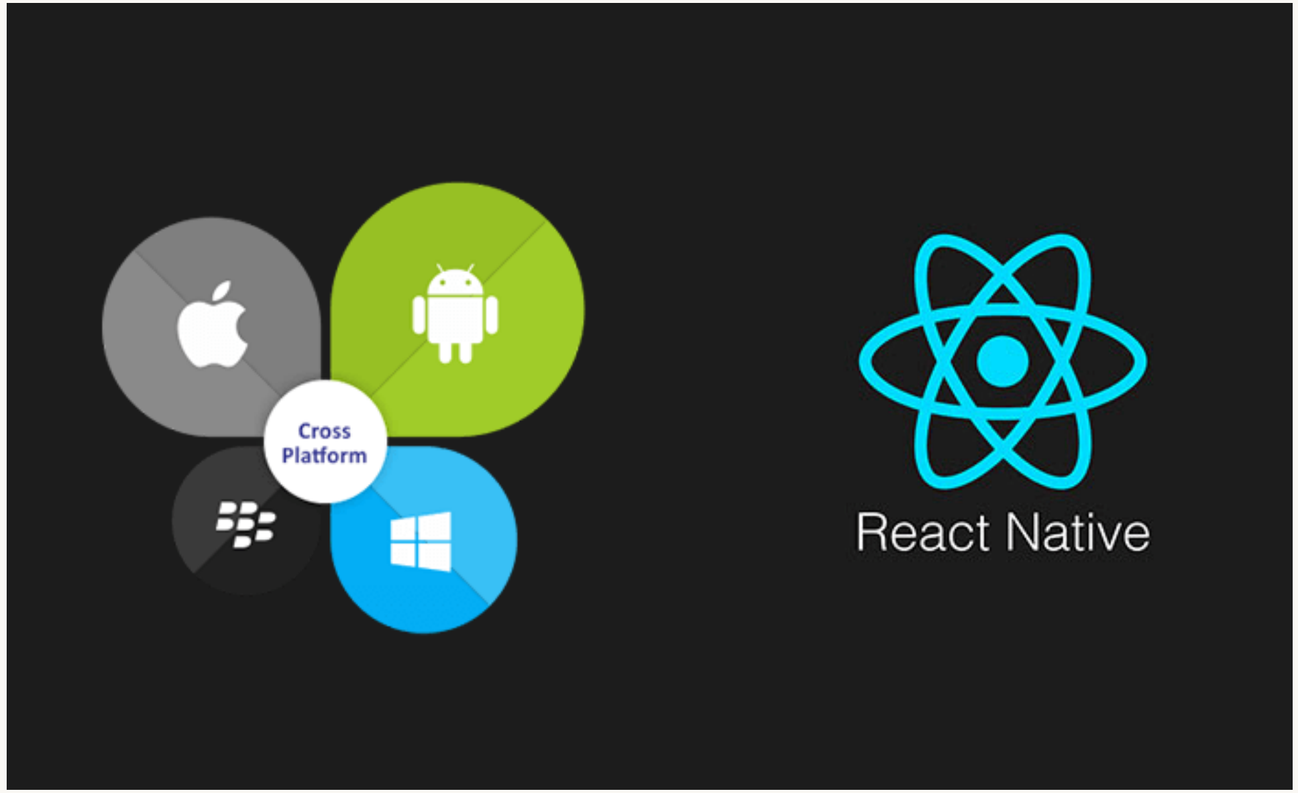
Introduction of React Native

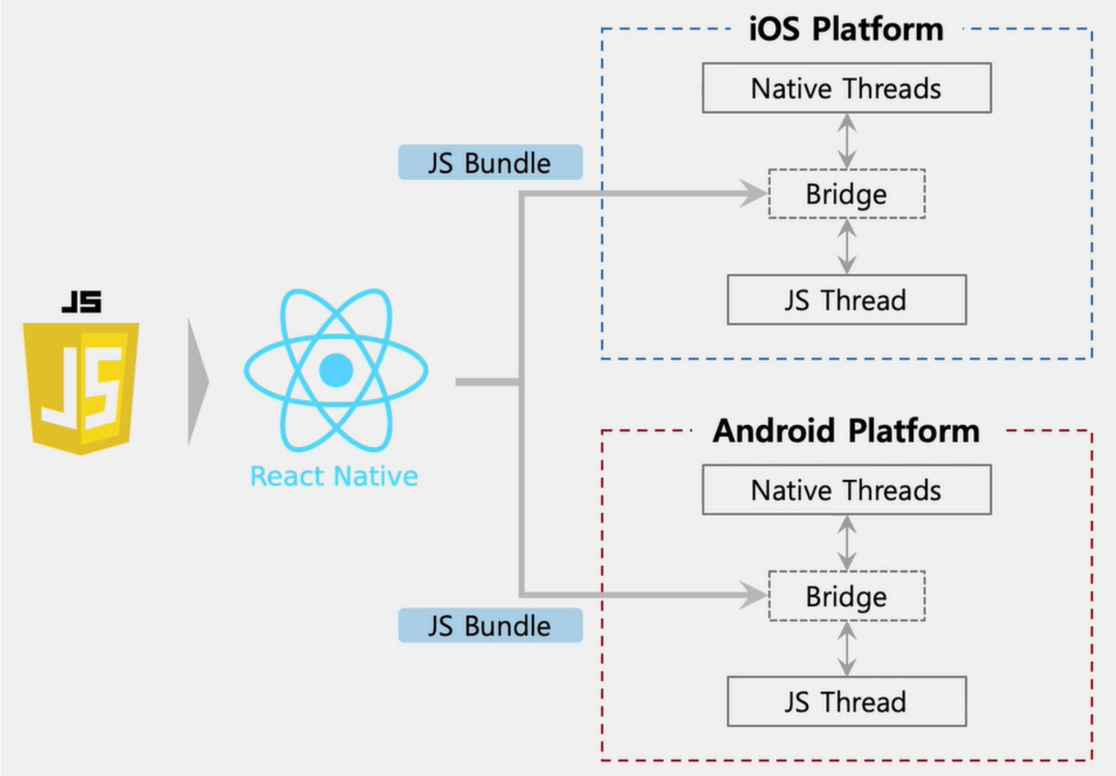
Contents.

01 What is React Native?

- Facebook이 개발한 Open Source Mobile Application Framework
- Javascript 하나로 Android, iOS, Web 대응



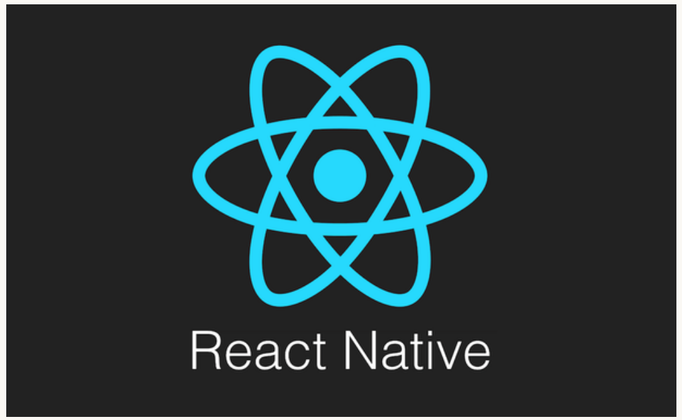
01 React Native 동작 원리



- Bridge의 역할: 각 플랫폼의 Native UI의 요소에 접근하는 인터페이스를 제공
- 즉, Component (Javascript)가 React Native에 전달하여 Bridge를 통해 각 플랫폼의 UI가 그려짐

01 React Native의 장점

하나의 코드로 관리,
러닝커브 높은 React 사용



코드푸시로 빠른 업데이트 ->
비용 절감

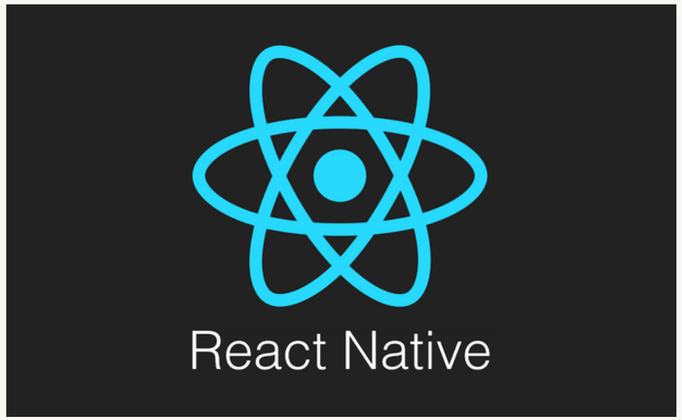
Fast Refresh

오픈소스 플랫폼

01 React Native의 단점

일부 기능
Native 접근 필요

오픈소스, 라이브러리 의존도



성능

잦은 업데이트

02 Expo CLI와 React Native CLI의 개념 및 비교 설명

Expo CLI

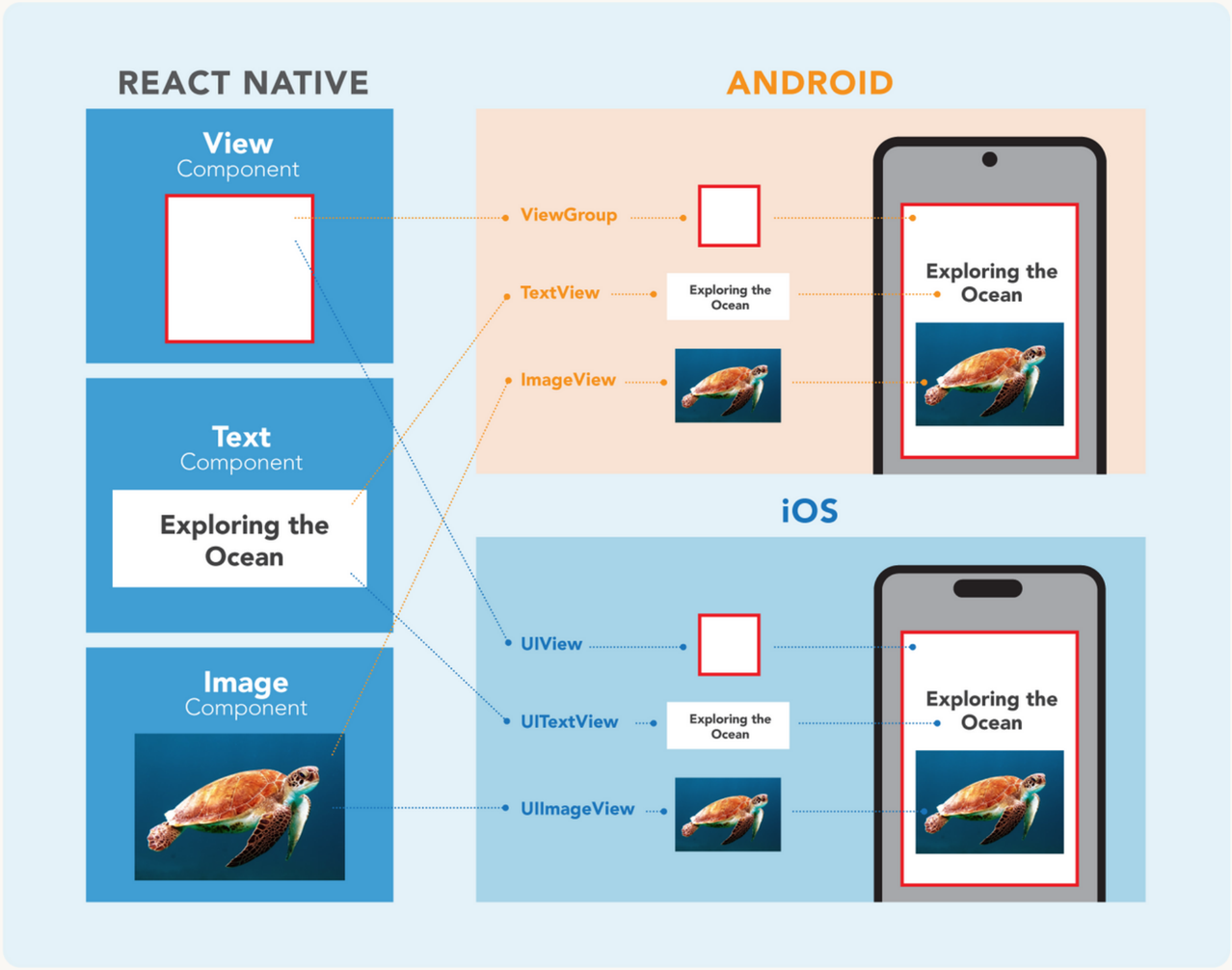
- React Native 앱을 쉽고 빠르게 구축 가능
- 장점
  - 기본제공되는 API, 라이브러리 -> 초반 앱 개발 단순화
  - Expo Go 어플만 있으면 기기 상관없이 프로젝트 실행 가능
- 단점
  - 제공되는 API만 사용가능
  - 추가 Native 모듈 사용 불가

02 Expo CLI와 React Native CLI의 개념 및 비교 설명

React Native CLI

- 고도화된 기능 개발 및 높은 개발 자유도
- 장점
  - Native 모듈 연결 가능
  - 다양한 라이브러리 사용으로 높은 자유도
- 단점
  - 기본 제공되는 라이브러리가 적어, 필요한 것이 있다면 직접 설치
  - Mac 개발 필수, Native 폴더 구조에 대한 기본 지식 필요

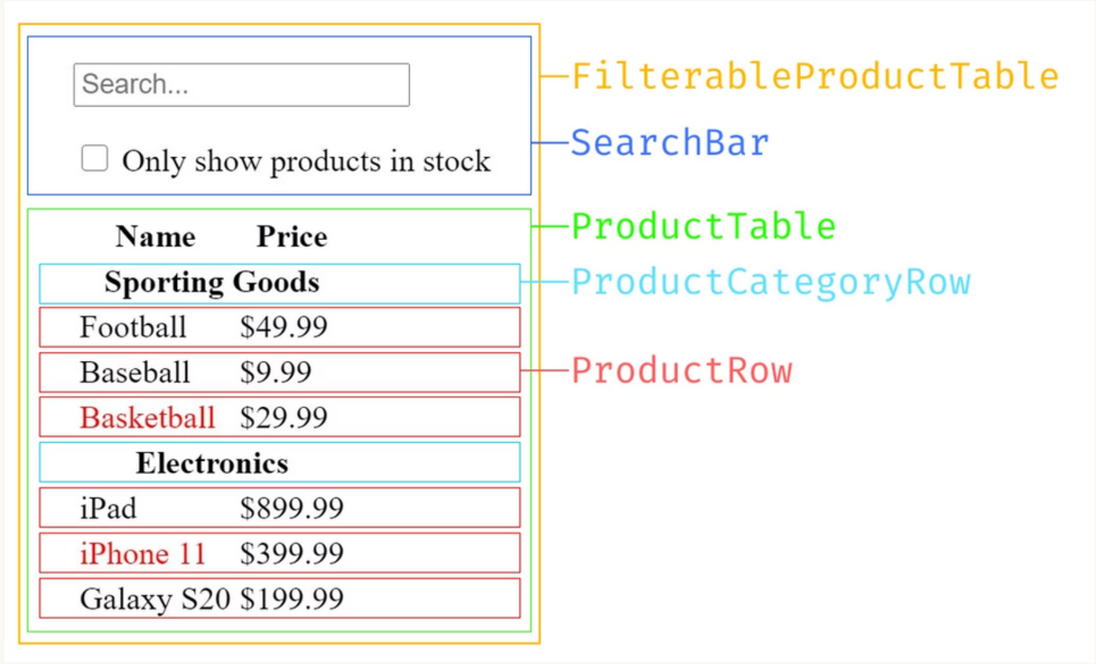
03 Core Components



04 Component 설명

Component

- 개발되어야하는 화면을 보았을 때 이를 어떠한 조각들(Components)로 나눌 수 있고 어떻게 합칠지 정확하게 파악하는 것이 중요



04 Component의 종류

Class형

- class 키워드 필요
- Component로 상속 받아야 함
- render() 메소드 반드시 필요
- state, lifeCycle 관련 기능 사용 가능
- 함수형보다 메모리 자원을 더 사용

```
import React from 'react';
import { Text, View } from 'react-native';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: 0};
    setInterval(() => {
      this.setState({ value: this.state.value + 1 });
    }, 1000);
  }

  render() {
    return (
      <View>
        <Text>{'State Value => ' + this.state.value}</Text>
      </View>
    );
  }
}
export default App;
```

04 Component의 종류

함수형

- state, lifeCycle 관련 기능 사용 불가능 -> hook으로 해결
- Class형보다 메모리 자원을 덜 사용
- Component 선언이 편함
- 공식문서에서도 함수형 Component + hook 사용을 권장

```jsx
import React from 'react';
import { Text, View } from 'react-native';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: 0};
    setInterval(() => {
      this.setState({ value: this.state.value + 1 });
    }, 1000);
  }

  render() {
    return (
      <View>
        <Text>{'State Value => ' + this.state.value}</Text>
      </View>
    );
  }
}
export default App;
```

05 React Hook

useEffect

- side effects를 수행하기 위한 훅으로, 데이터 가져오기, 구독 설정하기, 수동으로 React component의 DOM을 수정하는 작업 등을 수행할 때 유용

side effect

- 외부 세계와의 상호작용이나 가변 데이터의 변경 등을 포함하는 코드를 의미

useState

- 상태(state)를 관리할 수 있음
- ex) setText, setName, etc
- ex) useState(‘’), useState(0), useState(false)

```jsx
import React, { useState, useEffect } from 'react';
import { View, Text } from 'react-native';

const DataFetchingComponent = () => {
    const [data, setData] = useState(null);

    useEffect(() => {
        fetch('https://api.example.com/data')
            .then(response => response.json())
            .then(data => setData(data))
            .catch(error => console.error('Error fetching data:', error));
    }, []); // 빈 배열은 컴포넌트가 마운트될 때만 effect를 실행하도록 합니다.

    return (
        <View>
            {data ? <Text>{data}</Text> : <Text>Loading...</Text>}
        </View>
    );
};
```

06 Class Component의 Lifecycle