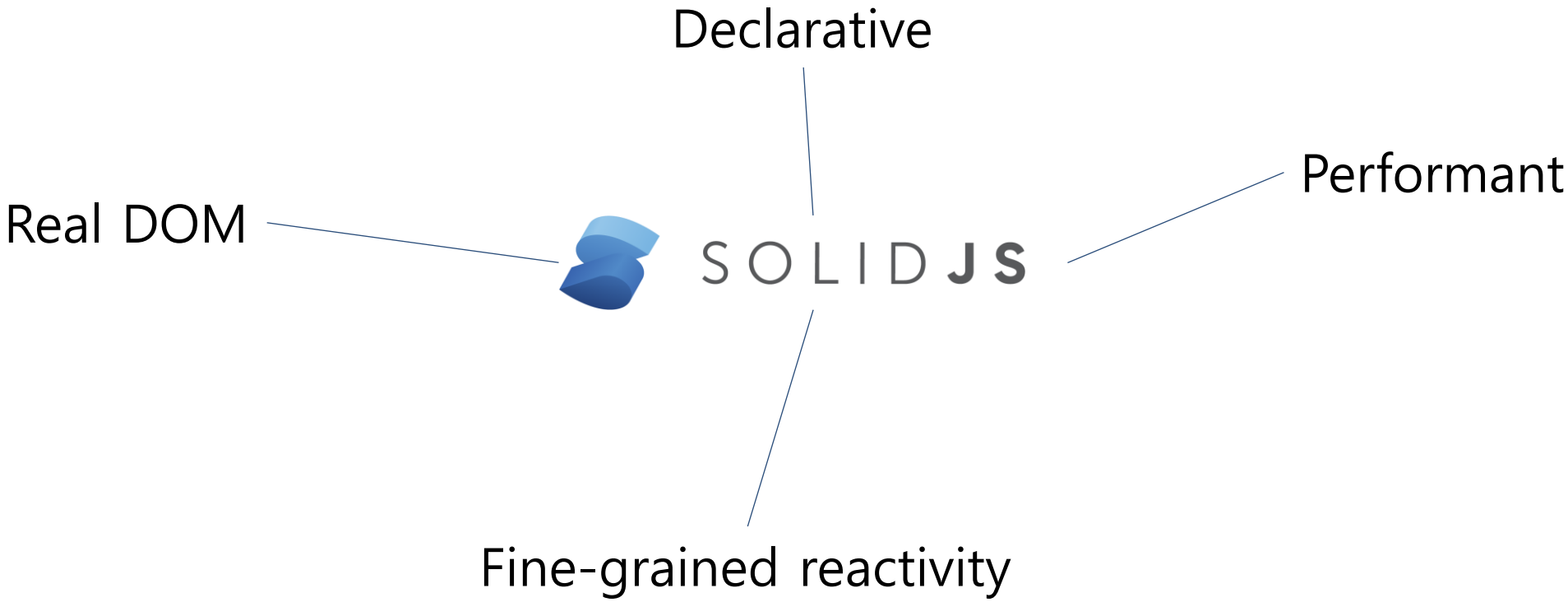
신세대 웹 UI 라이브러리, SolidJS

2016010873 박정욱

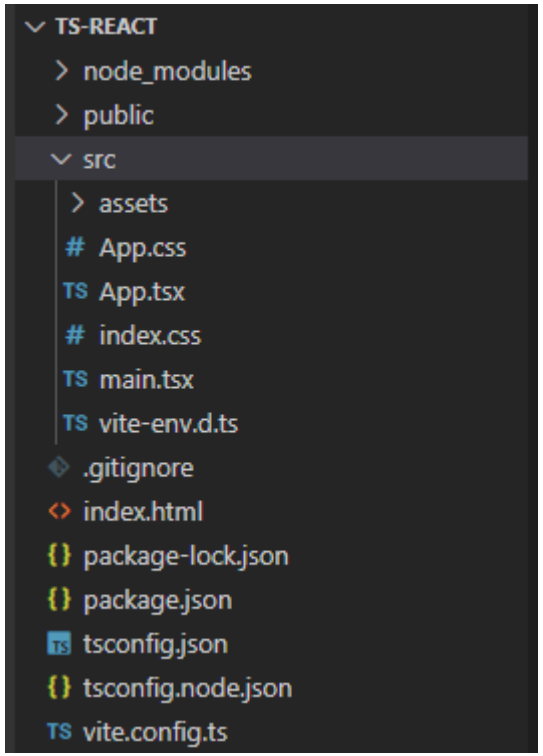
목차

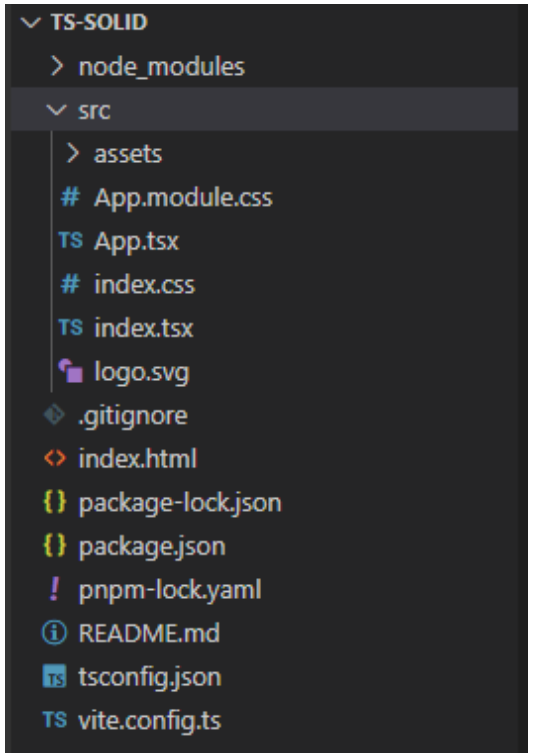
- SolidJS?
  - 타 라이브러리 대비 성능
- 반응형 프로그래밍의 핵심 키워드 2가지
- React의 단점
  - React의 스케줄링
  - Virtual DOM
- SolidJS의 장점
  - 세분화된 반응성

SolidJS?


Declarative
Performant
Real DOM
SOLIDJS
Fine-grained reactivity

SolidJS?





React와 비슷한 구조

SolidJS?

```tsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
)
```

```tsx
/* @refresh reload */
import { render } from 'solid-js/web';

import './index.css';
import App from './App';

render(() => <App />, document.getElementById('root') as HTMLElement);
```

React와 비슷한 구조

SolidJS?

```jsx
function App() {
  return (
    <div className="App">
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src="/vite.svg" className="logo" alt="Vite logo" />
        </a>
        <a href="https://reactjs.org" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </div>
  )
}
```

```jsx
const App: Component = () => {
  return (
    <div class={styles.App}>
      <header class={styles.header}>
        <img src={logo} class={styles.logo} alt="logo" />
        <p>
          Edit <code>src/App.tsx</code> and save to reload.
        </p>
        <a
          class={styles.link}
          href="https://github.com/solidjs/solid"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn Solid
        </a>
      </header>
    </div>
  );
};
```

React와 비슷한 구조

타 라이브러리 대비 성능

Name Duration for...	vanillajs-1	vanillajs	vanillajs-wc	solid-v1.4.4	vue-v3.2.37	svelte-v3.48.0	preact-v10.7.3	elm-v0.19.1-3	angular-v13.0.0	react-v17.0.2	react-mobX-v17.0.1 + 5.15.4	knockout-v3.5.0	ember-v4.2.0
Implementation notes	772	772	772										
create rows creating 1,000 rows (5 warmup runs).	36.3 ±0.3 (1.00)	36.7 ±0.2 (1.01)	40.5 ±0.2 (1.12)	37.6 ±0.2 (1.03)	45.6 ±0.4 (1.26)	48.2 ±0.3 (1.33)	46.4 ±0.3 (1.28)	50.2 ±1.9 (1.38)	46.1 ±0.5 (1.27)	48.8 ±0.2 (1.34)	49.6 ±0.2 (1.37)	78.0 ±0.4 (2.15)	100.7 ±0.3 (2.77)
replace all rows updating all 1,000 rows (5 warmup runs).	37.7 ±0.3 (1.00)	38.7 ±0.3 (1.03)	39.5 ±0.3 (1.05)	40.2 ±0.4 (1.07)	44.0 ±0.3 (1.17)	49.4 ±0.5 (1.31)	55.9 ±0.6 (1.48)	49.9 ±1.0 (1.32)	48.9 ±0.3 (1.30)	48.6 ±0.3 (1.29)	50.3 ±0.3 (1.33)	86.0 ±0.5 (2.28)	112.4 ±1.1 (2.98)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	88.9 ±1.4 (1.00)	93.7 ±2.2 (1.05)	93.8 ±2.4 (1.05)	92.9 ±1.6 (1.04)	107.0 ±3.4 (1.20)	97.7 ±1.2 (1.10)	116.5 ±2.5 (1.31)	101.0 ±2.5 (1.14)	94.1 ±1.8 (1.06)	123.4 ±3.0 (1.39)	110.7 ±2.0 (1.25)	99.9 ±2.4 (1.12)	113.9 ±1.7 (1.28)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	10.3 ±0.4 (1.02)	10.1 ±0.6 (1.00)	10.5 ±0.7 (1.04)	12.5 ±0.8 (1.23)	18.9 ±1.0 (1.86)	16.7 ±0.7 (1.65)	27.7 ±0.8 (2.73)	18.1 ±0.8 (1.78)	17.5 ±0.7 (1.73)	36.7 ±1.1 (3.62)	28.1 ±0.8 (2.77)	52.7 ±1.1 (5.19)	45.9 ±1.8 (4.52)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	25.3 ±0.6 (1.00)	26.4 ±0.7 (1.04)	25.7 ±0.6 (1.02)	26.5 ±0.5 (1.05)	27.7 ±0.5 (1.10)	27.4 ±0.7 (1.08)	30.2 ±0.5 (1.20)	41.4 ±4.5 (1.64)	167.7 ±0.9 (6.64)	164.6 ±1.2 (6.51)	159.7 ±0.7 (6.32)	30.7 ±0.4 (1.22)	35.1 ±0.4 (1.39)
remove row removing one row. (5 warmup runs).	11.8 ±2.3 (1.08)	10.9 ±0.4 (1.00)	12.2 ±2.1 (1.13)	16.5 ±1.9 (1.52)	12.0 ±0.4 (1.10)	12.1 ±1.8 (1.11)	17.1 ±0.1 (1.58)	26.3 ±2.7 (2.42)	11.2 ±2.2 (1.03)	12.7 ±0.4 (1.17)	15.1 ±1.2 (1.39)	13.4 ±1.4 (1.23)	12.8 ±0.2 (1.18)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	389.3 ±1.0 (1.01)	385.2 ±1.1 (1.00)	416.9 ±1.4 (1.08)	409.7 ±1.1 (1.06)	483.4 ±4.2 (1.25)	513.1 ±4.1 (1.33)	494.0 ±2.6 (1.28)	484.2 ±4.8 (1.26)	486.9 ±2.6 (1.26)	654.7 ±1.3 (1.70)	672.3 ±2.8 (1.75)	742.8 ±1.6 (1.93)	1,683.5 ± 248.3 (4.37)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown.	85.6 ±0.3 (1.02)	84.3 ±0.3 (1.00)	89.8 ±0.4 (1.07)	88.5 ±0.3 (1.05)	95.7 ±0.4 (1.14)	110.6 ±0.3 (1.31)	110.1 ±0.5 (1.31)	106.5 ±1.1 (1.26)	105.4 ±0.7 (1.25)	116.9 ±0.4 (1.39)	124.2 ±0.5 (1.47)	162.5 ±1.2 (1.93)	217.6 ±0.6 (2.58)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown. (5 warmup runs).	25.6 ±0.7 (1.00)	26.6 ±0.4 (1.04)	26.2 ±0.6 (1.03)	29.7 ±0.4 (1.16)	31.3 ±0.9 (1.23)	38.6 ±0.4 (1.51)	34.2 ±0.6 (1.34)	45.1 ±4.5 (1.77)	63.6 ±1.2 (2.49)	32.8 ±0.4 (1.29)	39.0 ±0.5 (1.53)	92.5 ±1.1 (3.62)	74.2 ±0.6 (2.91)
geometric mean of all factors in the table	1.01	1.02	1.06	1.13	1.24	1.29	1.45	1.51	1.63	1.80	1.82	2.03	2.39
compare: Green means significantly faster, red significantly slower	compare	compare	compare	compare	compare	compare	compare	compare	compare	compare	compare	compare	compare

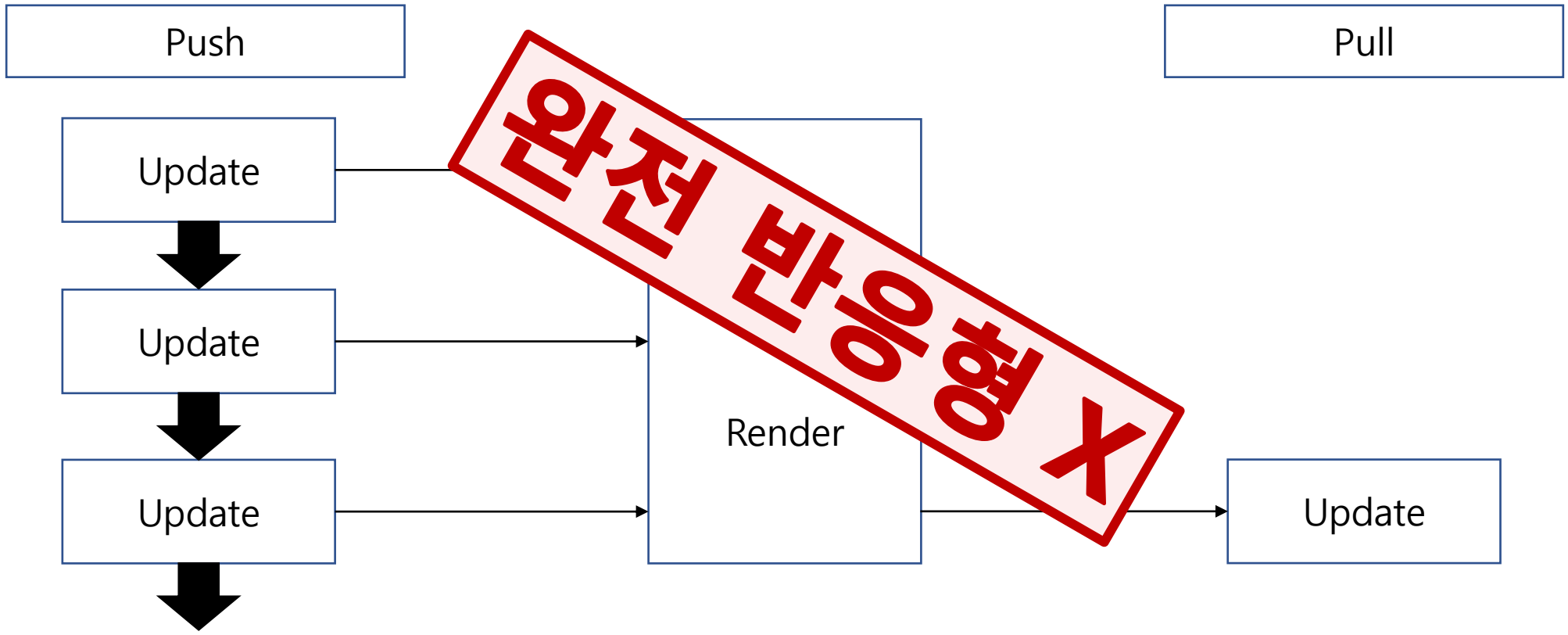
반응형 프로그래밍의 핵심 키워드 2가지

~~How~~ What

선언형 프로그래밍

데이터 중심 이벤트 방출

React의 스케줄링



변경 사항 존재 시 Push - React가 원할 때 한꺼번에 Pull

Virtual DOM

- 중요 렌더링 경로에서, Layout 과정과 Paint 과정은 ‘비싼’ 작업
- DOM과 CSSOM이 변경되면 두 작업을 다시 진행해야 함
  - Repainting: 요소의 스타일이 변경될 시 다시 그려야 함
  - Reflow(Re-layout): 요소의 위치/크기가 변경될 시 다시 계산해야 함
- 이러한 이유 때문에, React는 ‘Virtual DOM’ 개념을 도입함

Virtual DOM

- 웹 페이지 변경 = DOM/CSSOM **트리 변환**
- 기존 트리와 변경된 트리 간의 **트리 편집 거리**를 구하고 싶다!
  - 하나의 트리를 다른 트리로 변환하기 위해 필요한 연산의 최소 비용
- 순서 트리 T_1과 T_2에 대해…
- 트리 편집 거리 문제의 시간 복잡도는 $O(|T_1|^2|T_2|\log|T_2|)$

Virtual DOM

- 웹 페이지 변경 = DOM/CSSOM **트리 변환**
- 기존 트리 T_1과 변경된 트리 T_2 간의 **트리 편집 거리**?
  - 하나의 트리를 다른 트리로 변환하기 위해 필요한 연산의 최소 비용
- 트리 편집 거리 알고리즘 중 가장 효율적인 시간 복잡도는 $O(|T_1|^2|T_2|\log|T_2|)$
- React에서는 휴리스틱을 사용해 시간 복잡도 $O(n)$의 특수한 알고리즘을 고안
- 이렇게 찾은 변경점을 Virtual DOM에 push
- 이후 변경된 Virtual DOM을 React가 pull하여 실제 DOM을 변경

세분화된 반응성

- Virtual DOM이 있어야 하는 이유?
  -> 변경이 생길 때마다 비교 알고리즘 수행
  -> 스케줄링을 통한 일괄 처리로 성능을 높였지만 완벽하지 않음

- 하지만 값의 변화를 '구독'하여 추적할 수 있다면?

- 그리고 자신이 의존하는 값에 대한 정보를 관리할 수 있다면?

세분화된 반응성

Multiplier.tsx

src > Multiplier.tsx > ...

```tsx
import { useEffect, useState } from "react";

export const Multiplier = () => {
    const [operand1, setOperand1] = useState(0);
    const [operand2, setOperand2] = useState(0);

    const result = operand1 * operand2;

    useEffect(() => {
        console.log(`Operand 1 has been changed into ${operand1}!`);
    }, [operand1]);

    useEffect(() => {
        console.log(`Operand 2 has been changed into ${operand2}!`);
    }, [operand2]);

    useEffect(() => {
        console.log(`Result has been changed into ${result}!`);
    }, [result]);

    return (
    <>
    <h1>{operand1} * {operand2} = {result}</h1>
    <button onClick={() => setOperand1(operand1 + 1)}>
        Click me to increment operand 1!
    </button>
    <button onClick={() => setOperand2(operand2 + 1)}>
        Click me to increment operand 2!
    </button>
    </>
    );
};
```

- React의 상태 관리 기본 요소 Hook
  - `useState()`, `useEffect()`, `useMemo()`, etc....
- 특정 Component와 관련된 hook은 해당 Component 안에서만 관리

= 상태가 늘어날수록 코드 길이 ↑

세분화된 반응성

Multiplier.tsx

```tsx
import { useEffect, useState } from "react";

export const Multiplier = () => {
    const [operand1, setOperand1] = useState(0);
    const [operand2, setOperand2] = useState(0);

    const result = operand1 * operand2;

    useEffect(() => {
        console.log(`Operand 1 has been changed into ${operand1}!`);
    }, [operand1]);

    useEffect(() => {
        console.log(`Operand 2 has been changed into ${operand2}!`);
    }, [operand2]);

    useEffect(() => {
        console.log(`Result has been changed into ${result}!`);
    }, [result]);

    return (
    <>
    <h1>{operand1} * {operand2} = {result}</h1>
    <button onClick={() => setOperand1(operand1 + 1)}>
        Click me to increment operand 1!
    </button>
    <button onClick={() => setOperand2(operand2 + 1)}>
        Click me to increment operand 2!
    </button>
    </>
    );
};
```

- `useEffect()`의 경우 dependency array를 제공하여, array의 요소값이 변경되면 재렌더링함
- 관리할 상태값이 늘어날수록, dependency array 또한 관리하기 어려워짐

세분화된 반응성

Multiplier.tsx > ...

```tsx
import { Component } from "solid-js";
import * as signals from "./Signals";

export const Multiplier: Component = () => {
    return (
    <>
    <h1>{signals.operand1()} * {signals.operand2()} = {signals.result()}</h1>
    <button onClick={() => signals.setOperand1(signals.operand1() + 1)}>
        Click me to increment operand 1!
    </button>
    <button onClick={() => signals.setOperand2(signals.operand2() + 1)}>
        Click me to increment operand 2!
    </button>
    </>
    );
};
```

Signals.ts > ...

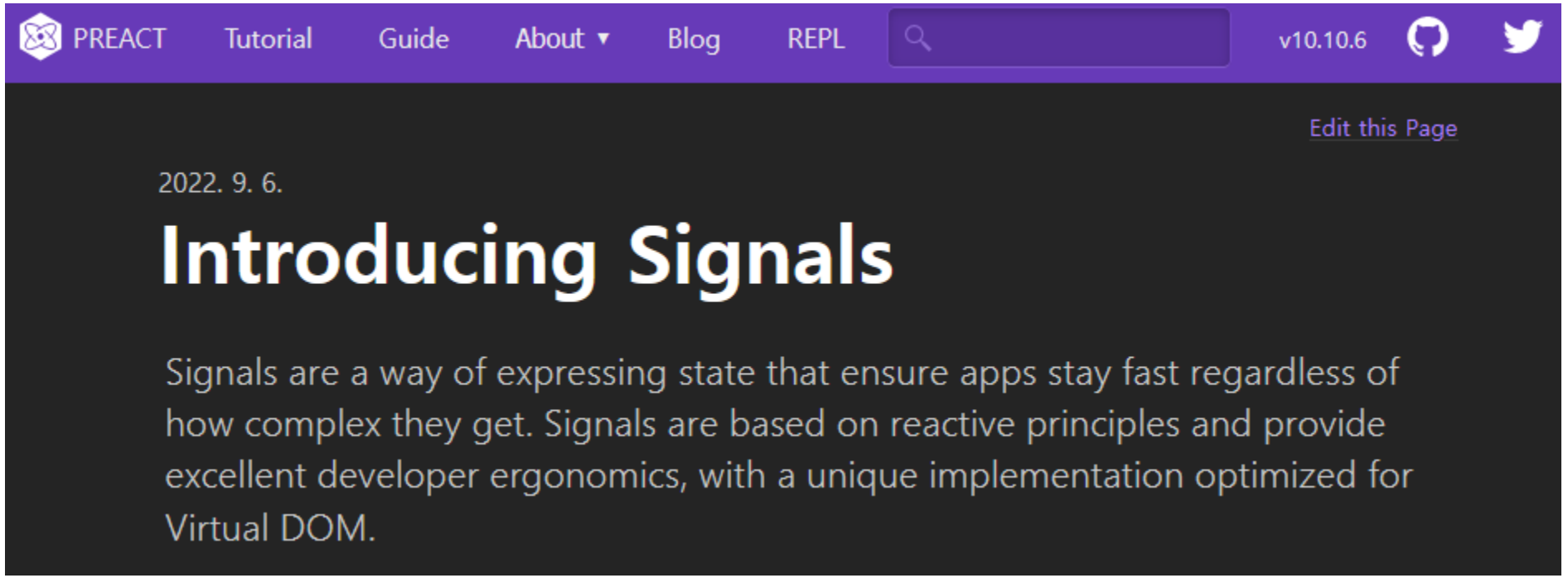
```ts
import { createSignal, createEffect } from "solid-js";

export const [operand1, setOperand1] = createSignal(0);
export const [operand2, setOperand2] = createSignal(0);

export const result = () => operand1() * operand2();

createEffect(() => {
    console.log(`Operand 1 has been changed into ${operand1()}!`);
});

createEffect(() => {
    console.log(`Operand 2 has been changed into ${operand2()}!`);
});

createEffect(() => {
    console.log(`Result has been changed into ${result()}!`);
});
```

- Solid는 상태값을 별도로 분리하여 관리할 수 있음
- **각각의 상태값이 별도로 추적**되어, dependency array는 불필요
- 프로젝트의 크기가 커질수록, 이런 장점은 극대화됨

정리

- React는 설계상 완전히 반응형 X
- 트리 편집 거리 알고리즘은 원래 엄청 느림
- Re-layout과 Repaint 또한 엄청 느림
- SolidJS는 '세분화된 반응성'을 통해 React의 성능을 뛰어넘음

...인 줄 알았으나



발표 날짜 하루 전, React의 경량판인 Preact에서 Signal 도입 발표
(물론 React에서도 사용 가능)