

# Angular, React?

두 가지 관점에서 바라본 Angular와 React

# Contents


---

1. Angular, React 의 목적
2. 첫 번째 관점  
Framework vs. Library
3. 두 번째 관점  
DOM 조작 방식과 Data binding
4. React와 D3.js의 결합성

# Angular, React 의 목적

## Angular란?

이 문서는 Angular를 이해하기 위한 기본 내용을 다룹니다: Angular는 무엇인지, 사용하면 어떤 점이 좋은지, 애플리케이션 개발을 어떻게 시작할 수 있는지 확인해 보세요.

Angular는 TypeScript 를 기반으로 개발된 개발 플랫폼입니다. 플랫폼이면서 동시에:

- 확장가능한 컴포넌트 구조로 웹 애플리케이션을 만드는 프레임워크입니다.
- 라우팅, 폼 관리, 클라이언트-서버 통신 등 웹 개발에 필요한 라이브러리를 조화롭게 통합한 모음집입니다.
- 애플리케이션 개발, 빌드, 테스트, 수정에 필요한 개발자 도구를 제공합니다.

거의 All-in-one 플랫폼

Angular는 혼자 개발하는 프로젝트는 물론이고 기업용 애플리케이션에도 활용할 수 있습니다. 그리고 최신 기술을 쉽게 도입할 수 있도록 설계되었기 때문에 적은 노력으로 큰 생산성 향상을 낼 수도 있습니다. 무엇보다도, Angular 생태계에 함께하는 개발자, 라이브러리 개발자, 콘텐츠 작성자는 총 170만명 이상에 달합니다.

이 문서에서 다루는 앱은 [라이브 예제 링크](#) / [다운로드 링크](#)에서 직접 실행하거나 다운받아 실행할 수 있습니다.

# Angular, React 의 목적

 React

문서 자습서 블로그 커뮤니티

🔍 검색

v18.2.0 🌐 Languages GitHub

# React

사용자 인터페이스를 만들기 위한 JavaScript 라이브러리

시작하기 자습서 읽어보기 >

## 선언형

React는 상호작용이 많은 UI를 만들 때 생기는 어려움을 줄여줍니다. 애플리케이션의 각 상태에 대한 간단한 뷰만 설계하세요. 그럼 React는 데이터가 변경됨에 따라 적절한 컴포넌트만 효율적으로 갱신하고 렌더링합니다.

선언형 뷰는 코드를 예측 가능하고 디버그하기 쉽게 만들어 줍니다.

## 컴포넌트 기반

스스로 상태를 관리하는 캡슐화된 컴포넌트를 만드세요. 그리고 이를 조합해 복잡한 UI를 만들어보세요.

컴포넌트 로직은 템플릿이 아닌 JavaScript로 작성됩니다. 따라서 다양한 형식의 데이터를 앱 안에서 손쉽게 전달할 수 있고, DOM과는 별개로 상태를 관리할 수 있습니다.

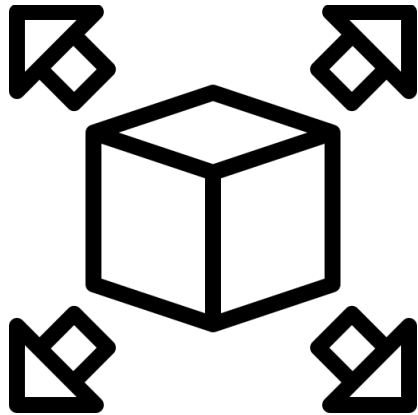
## 한 번 배워서 어디서나 사용하기

기술 스택의 나머지 부분에는 관여하지 않기 때문에, 기존 코드를 다시 작성하지 않고도 React의 새로운 기능을 이용해 개발할 수 있습니다.

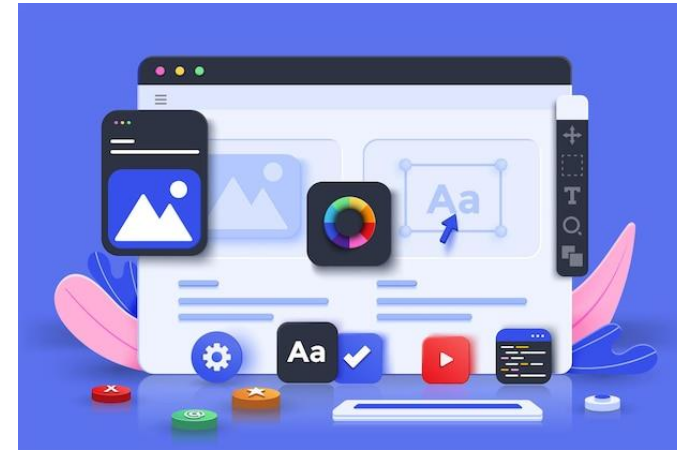
React는 Node 서버에서 렌더링을 할 수도 있고, React Native를 이용하면 모바일 앱도 만들 수 있습니다.

# Angular, React 의 목적

> 공통적인 특징



확장 가능한 컴포넌트 구조



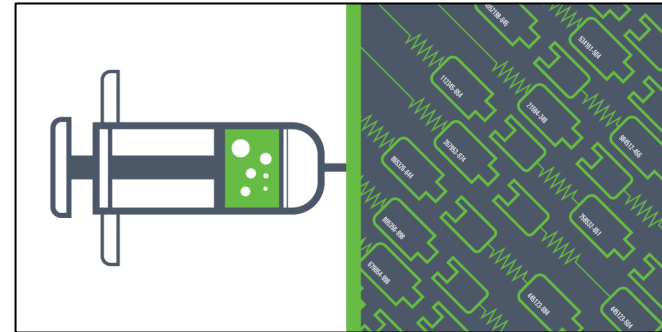
UI 제작을 위한 컴포넌트 제공

# Angular, React 의 목적

> 차이점



Typescript 기반 동작

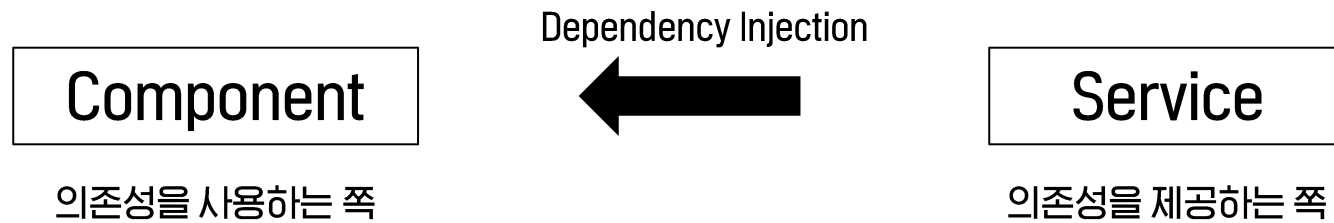


Dependency Injection 친화적

→ 컴포넌트에 필요한 인스턴스를 외부에서 받아옴

# Angular, React 의 목적

> 차이점



# Angular, React 의 목적

> 차이점



```
constructor(private ws: WorkspaceService) {  
  this.ws.selectedBusIDs.subscribe(b => {  
    this.selectedBusIDs = b;  
  });  
};
```

의존성 사용하는 Class 생성자에서 private으로 요청

```
@Injectable()
```

@Injectable() 데코레이터 사용

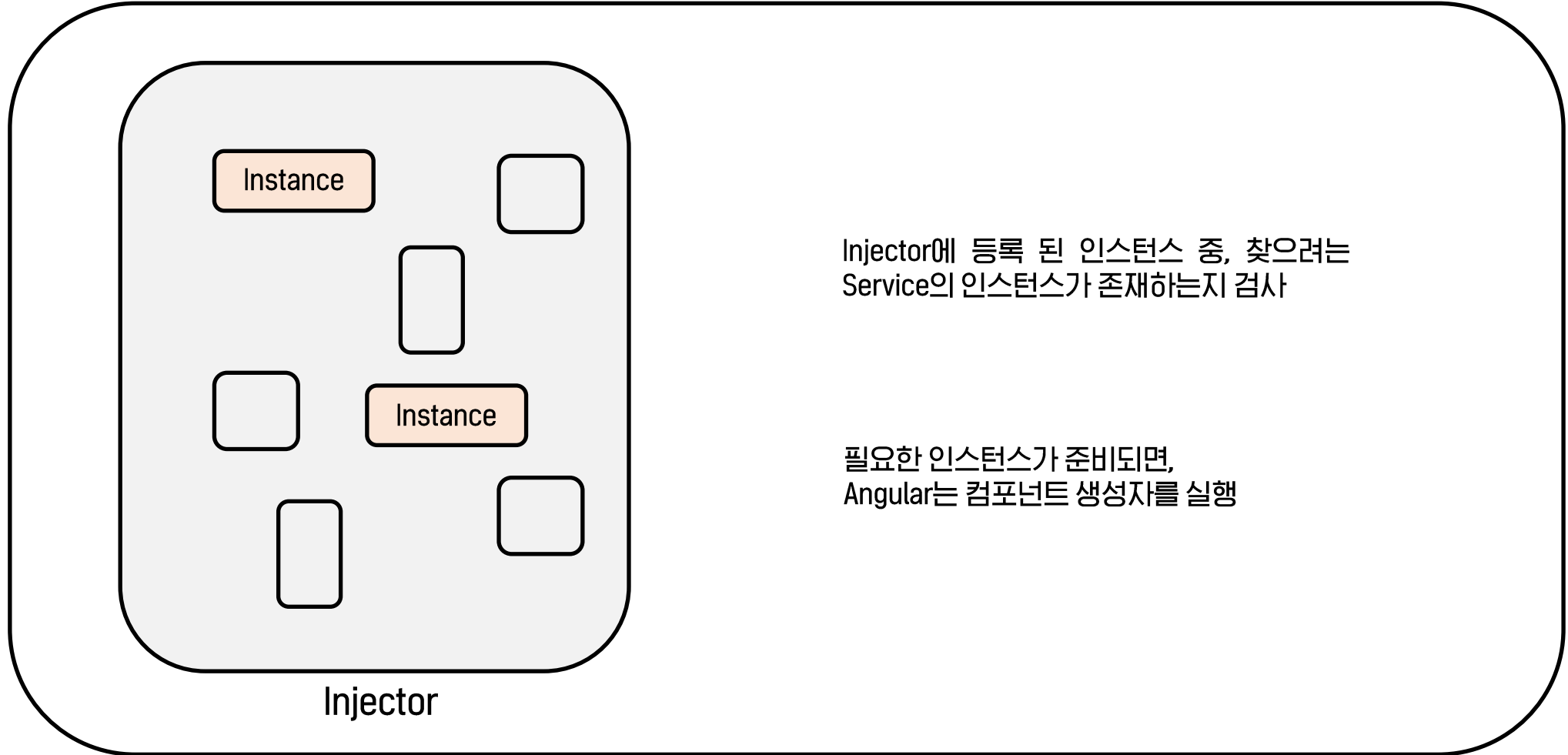


# Angular, React 의 목적

> 차이점

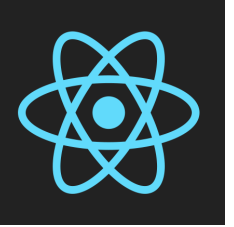


## Angular Framework



# Angular, React 의 목적

> 차이점



Javascript 기반 동작  
[Typescript 지원]

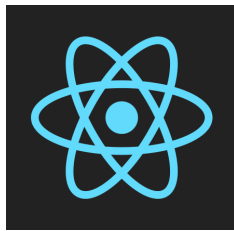
Dependency  
Injection?

Dependency Injection에 친숙하지 않음

→ 제대로 DI를 하려면?  
IoC 컨테이너가 필요함. (React에 임베딩 X)

# Angular, React 의 목적

> 차이점

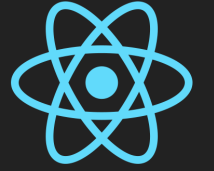


*Don't call us. We will call you.*  
– Hollywood principle

IoC : Inversion of Control (제어의 역전)

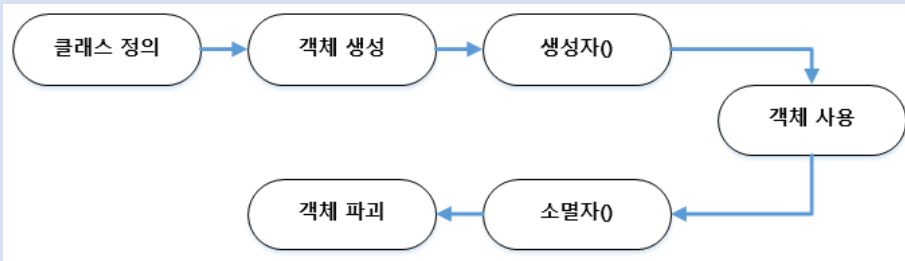
# Angular, React 의 목적

> 차이점



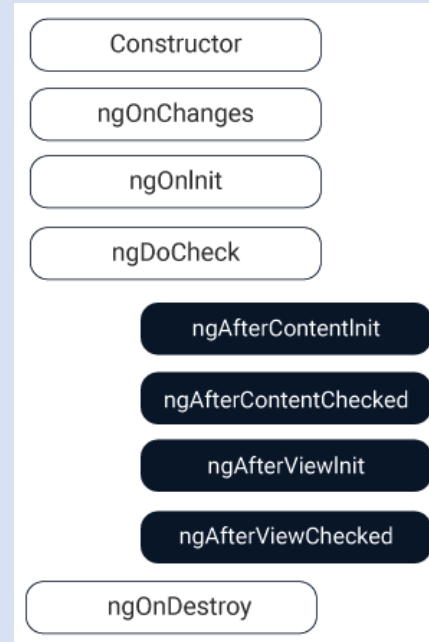
## IoC : Inversion of Control (제어의 역전)

개발자가 원하는 인스턴스, 메서드 제어를 외부에 위임하는 것



### Framework 없이 개발

-> 개발자가 직접 생명 주기를 관리해야 함



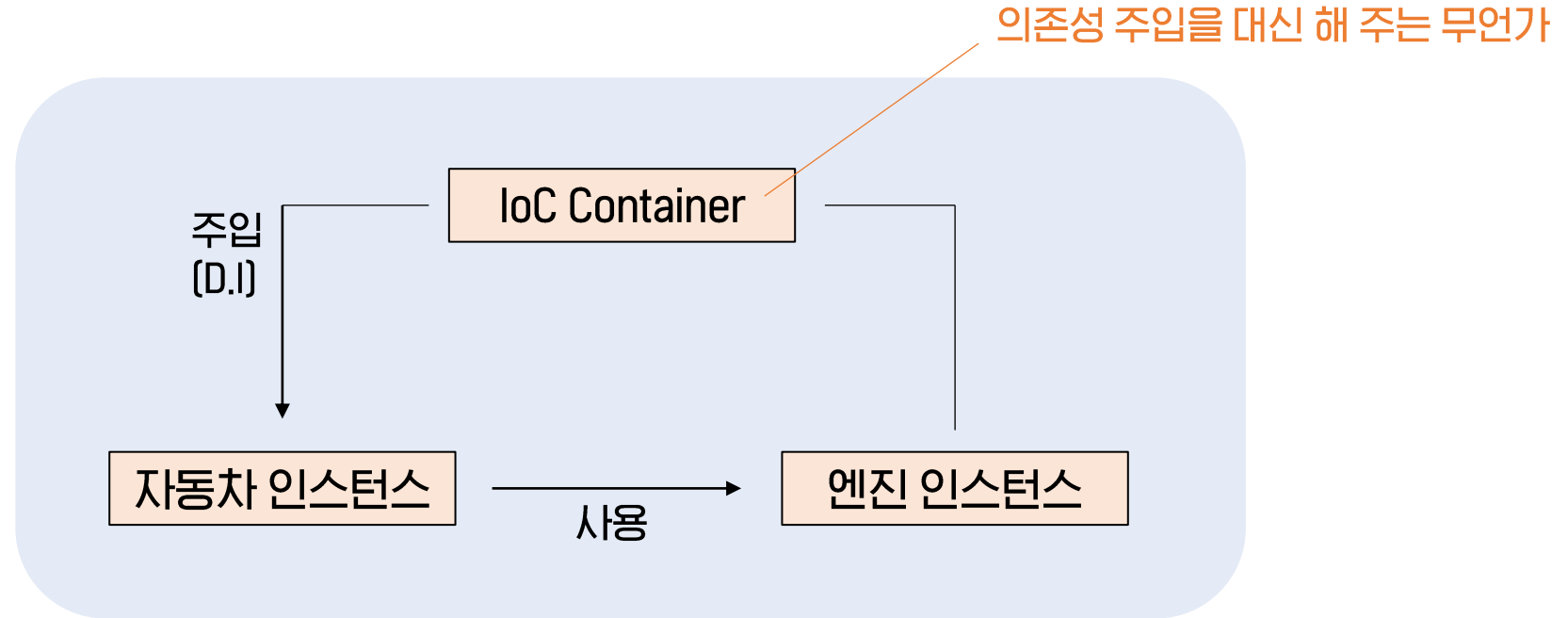
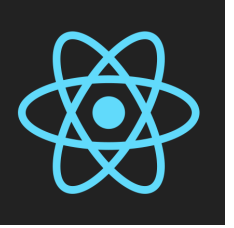
### Framework 로 개발

-> 프레임워크가 생명 주기를 관리함

-> 개발자는 규칙을 따르기만 하면 됨

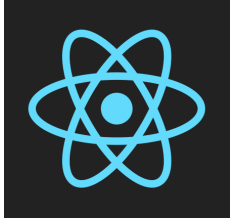
# Angular, React 의 목적

> 차이점



# Angular, React 의 목적

> 차이점



Angular : 인스턴스를 알아서 관리해 준다. 개발자는 규칙을 따를 뿐.

React : 개발자가 인스턴스 생명 주기를 좀 더 주도적으로 고려해야 한다.

-> Framework와 Library 의 차이

---

# 첫 번째 관점

> Framework vs Library

# 첫 번째 관점

> Framework vs Library



Frame[틀, 규칙] + work[일, software 작업]

Application 개발 시 필요한 기능들의 뼈대 제공

개발자는 정해진 틀(= 규칙)에 따라 구현

제어 흐름이 프레임워크에게 있음



# 첫 번째 관점

> Framework vs Library



Library : 도서관, 책들의 집합

여러 책들이 꽂혀 있고, 이용자는 필요한 책을 꺼내 사용하면 됨  
개발하기 위해 필요한 것들을 미리 구현 해 놓은 도구  
제어 흐름이 개발자의 Application에 있음

# 첫 번째 관점

> Framework vs Library



```

  ▾ bar-chart
    # bar-chart.component.css
    <> bar-chart.component.html
    TS bar-chart.component.spec.ts
    TS bar-chart.component.ts
  ▾ circle-diagram
    # circle-diagram.component.css
    <> circle-diagram.component.html
    TS circle-diagram.component.spec.ts
    TS circle-diagram.component.ts
  ▾ scatter-plot
    # scatter-plot.component.css
    <> scatter-plot.component.html
    TS scatter-plot.component.spec.ts
    TS scatter-plot.component.ts
  TS app-routing.module.ts
  # app.component.css
  <> app.component.html
  TS app.component.spec.ts
  TS app.component.ts
  TS app.module.ts

```

```

@Component({
  selector: 'app-scatter',
  templateUrl: './scatter-plot.component.html',
  styleUrls: ['./scatter-plot.component.css']
})
export class ScatterPlotComponent implements OnInit {
  private margin = 50;
  private width = 900 - (this.margin * 2);
  private height = 400 - (this.margin);
  constructor() { }
}

```

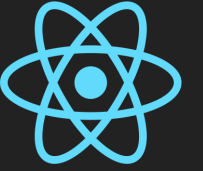
-> 개발자는 이 틀에 맞춰서 개발

ng generate component

-> 정해진 구조가 생성됨

# 첫 번째 관점

> Framework vs Library



```
const searchPage = () => {
  return (
    <s.PageTemplate>
      <SearchBar />
      <ContentWrapper contentName="최근 검색어">
        {recentKeywords.map((recentKeyword) => {
          return (
            <RecentSearched keyword={recentKeyword.keyword} articleId={recentKeyword.articleId} />
          );
        })}
      </ContentWrapper>
      <ContentWrapper contentName="인기 검색어">
        {popularKeywords.map((popularKeyword) => {
          return (
            <PopularKeyword
              rank={popularKeyword.rank}
              keyword={popularKeyword.keyword}
              articleId={popularKeyword.articleId}
            />
          );
        })}
      </ContentWrapper>
    </s.PageTemplate>
  );
};

export default searchPage;
```

```
const SearchBar = () => {
  return (
    <form action="/result" method="get">
      <s.Wrapper>
        <SearchIcon />
        <s.SearchInput type="text" name="search" placeholder="검색어를 입력해주세요." required />
      </s.Wrapper>
    </form>
  );
};

export default SearchBar;
```

DOM element를 custom하여 화면에 뿌린다.

-> Library를 사용하기 위한 제약은 어느 정도 있지만, 구성은 자유롭게 한다.

---

## 두 번째 관점

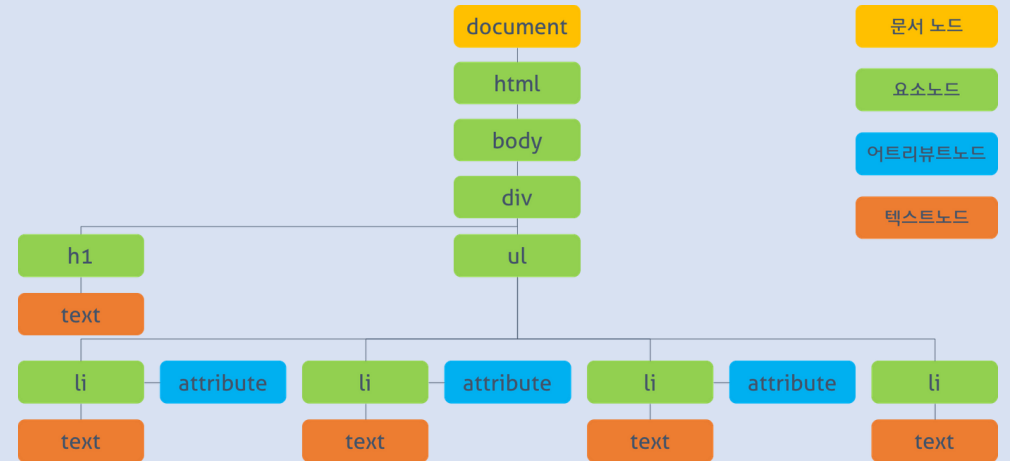
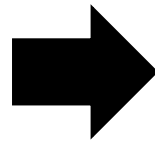
> DOM 조작 방식과 데이터 바인딩

# 두 번째 관점

> DOM 조작 방식과 데이터 바인딩



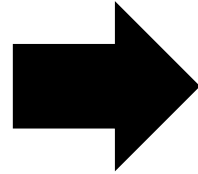
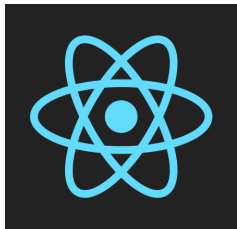
Web Page를 구성한다



DOM element와 구조를 구성한다

# 두 번째 관점

> DOM 조작 방식과 데이터 바인딩



More efficiently !

! UI 변경

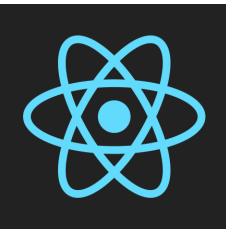


브라우저 렌더링

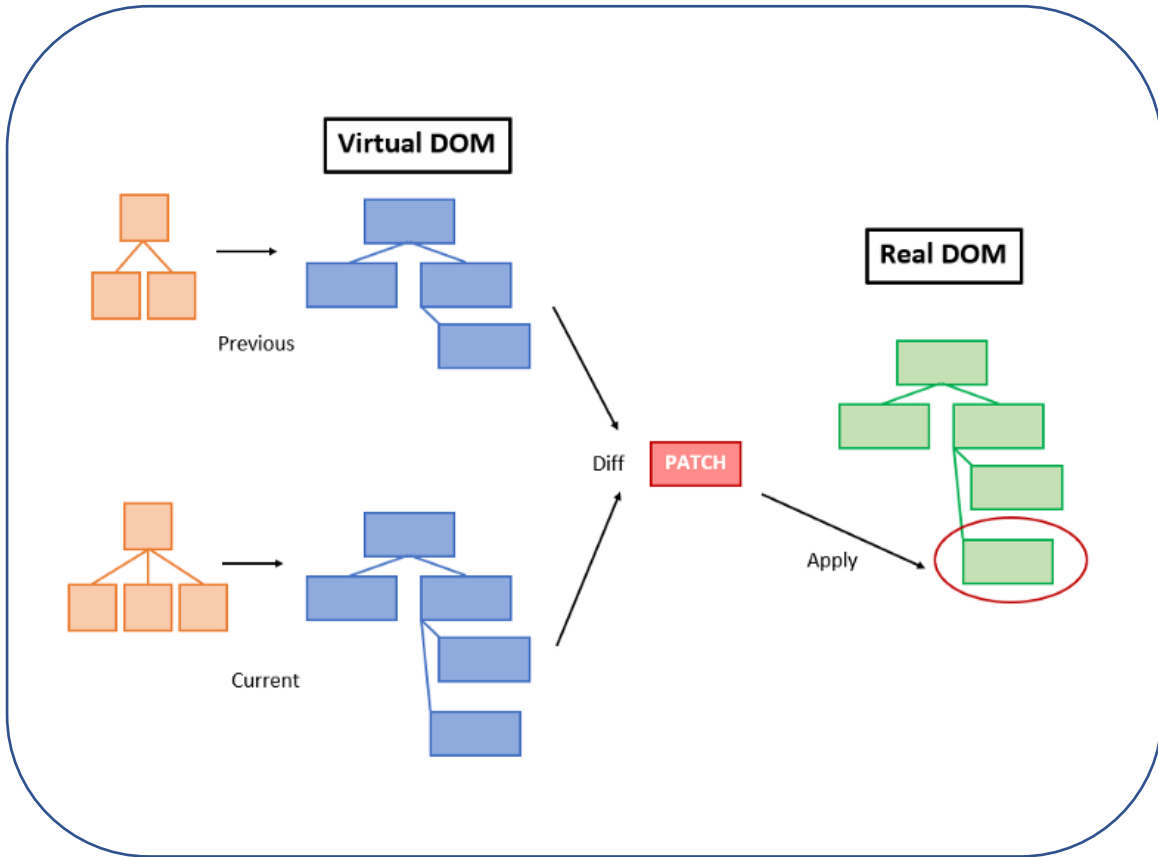
1. DOM Tree 생성
2. Render Tree 생성
3. Layout
4. Paint

# 두 번째 관점

> DOM 조작 방식과 데이터 바인딩



Virtual DOM



- UI가 변경되면, 전체 UI를 Virtual DOM으로 렌더링
- 현재 Virtual DOM과 이전 Virtual DOM을 비교
- 이때 굉장히 효율적인 diff 알고리즘 사용
- 변경된 부분만 실제 DOM에 반영

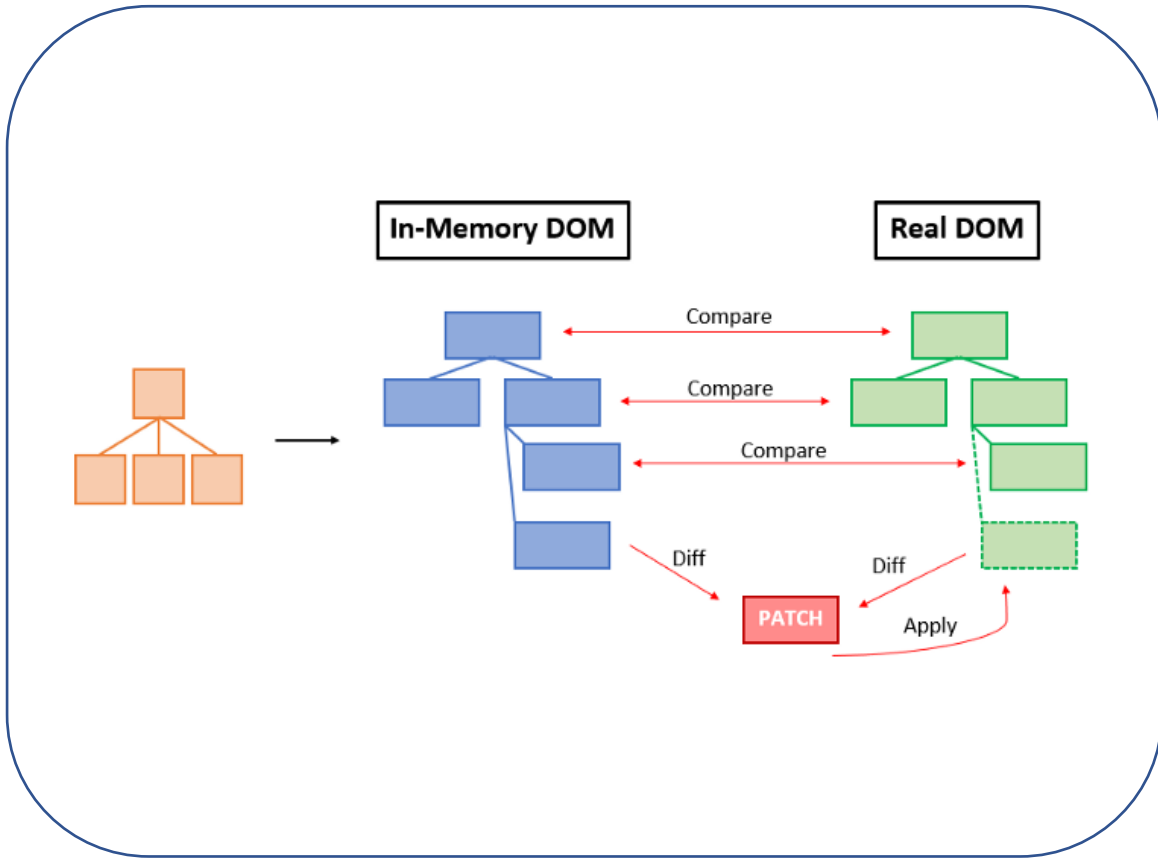
Virtual DOM은 UI의 가상적인 표현을 메모리에 저장하고,  
React DOM과 같은 라이브러리에 의해 실제 DOM과 동기화하는 프로그래밍 개념

# 두 번째 관점

> DOM 조작 방식과 데이터 바인딩



Incremental DOM



- 최초 렌더링 시, 모든 DOM element들은 명령어로 바뀜
- 이후 UI 변경 시, 바뀐 명령어만 메모리 상에서 찾을
- 바뀐 명령어만 실제 DOM에 반영



# 두 번째 관점

> DOM 조작 방식과 데이터 바인딩



Incremental DOM

```
<h1 class="header">HELLO WORLD</h1>
```

```
import ID from 'incremental-dom'

function renderHelloWorld() {
  ID.elementOpen('h1', null, ['class', 'header'], null);
  ID.text('HELLO WORLD');
  ID.elementClose('h1');
}

patch(document.body, renderHelloWorld);
```

- 최초 렌더링 시, 모든 DOM element들은 명령어로 바뀜
- 이후 UI 변경 시, 바뀐 명령어만 메모리 상에서 찾을
- 바뀐 명령어만 실제 DOM에 반영

Incremental DOM은 명령(instruction) 묶음을 통해 모든 컴포넌트를 컴파일한다.  
이 명령들은 DOM Tree를 생성하고, 변경점을 찾아낸다.

# 두 번째 관점

> DOM 조작 방식과 데이터 바인딩



1. Template reference variables
2. ElementRef

Angular에서 DOM을 조작하려면? @ViewChild / @ViewChildren



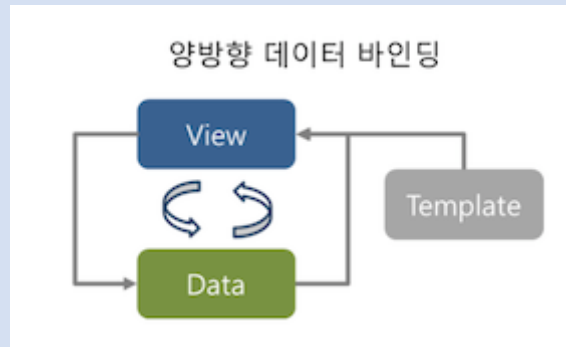
```
@ViewChild('rootSvg') rootSvg!: ElementRef;
```

Rendering logic과 Presentation logic을 구별할 필요성 = 바인딩

-> html 파일의 데이터와, 이에 접근하는 ts 파일의 데이터를 동기화해야 함

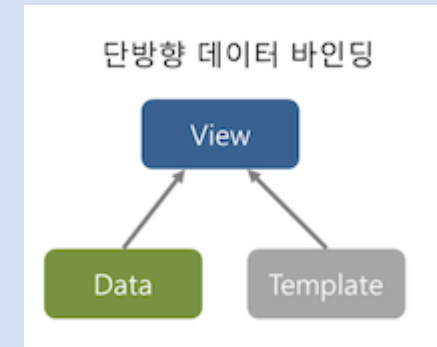
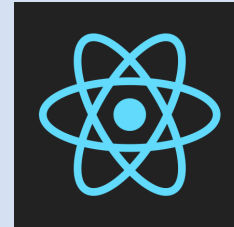
# 두 번째 관점

> DOM 조작 방식과 데이터 바인딩



## 양방향 데이터 바인딩

-> 별도의 함수 없이,  
html 파일의 데이터와 ts 파일의 데이터를 동기화 할 수 있는 경우



## 단방향 데이터 바인딩

-> 별도의 함수를 사용해야 하는 경우

---

# React와 D3의 결합성

# React와 D3의 결합성



같이 사용하면 (React 위에 D3를 올리면), 좋을까? 나쁠까?

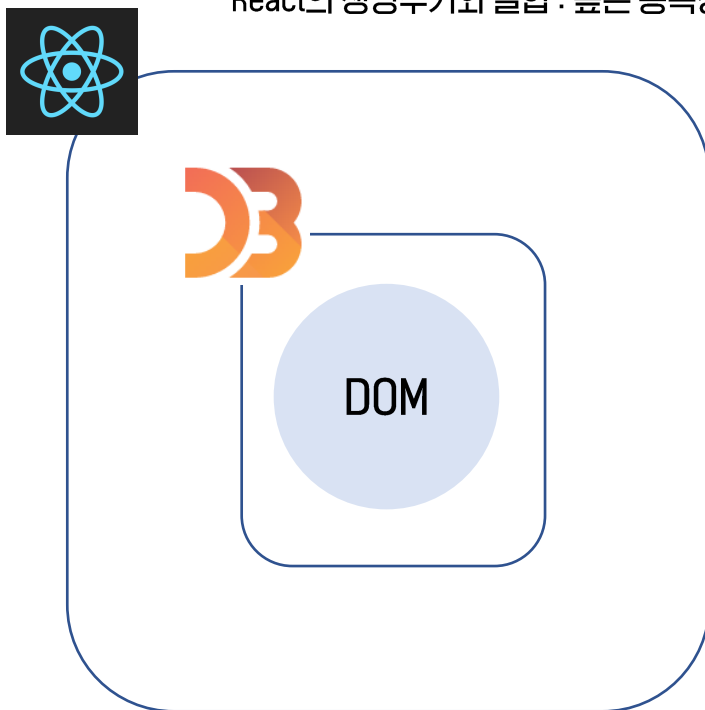
# React와 D3의 결합성

3가지 접근 방법

> 접근 방법

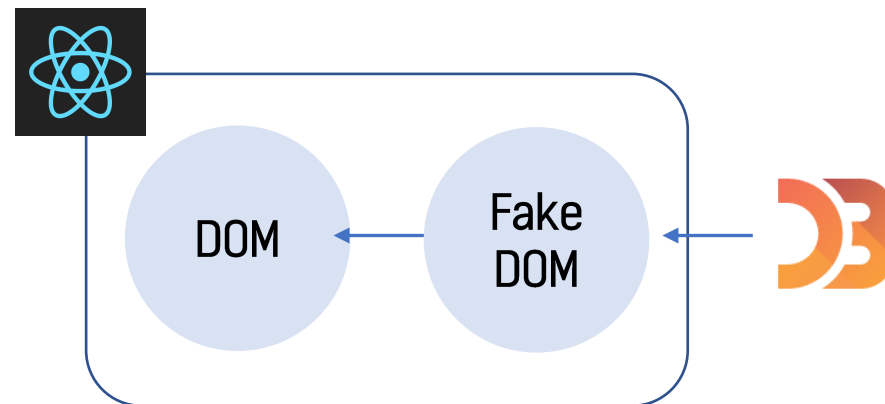
## 1. D3에 가능한 많은 DOM 제어권을 제공

React의 생명주기와 결합: 높은 종속성 이슈



## 2. Fake DOM 사용

React Faux DOM 라이브러리 추가 사용  
두 배의 DOM 가상화: 성능 이슈

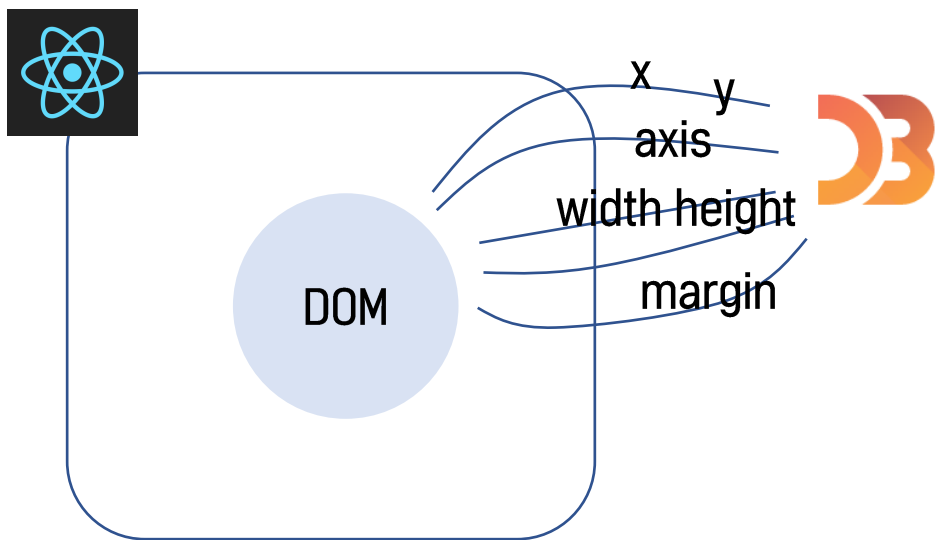


# React와 D3의 결합성

> 접근 방법

3가지 접근 방법

## 3. 수학적 계산은 D3가, 렌더링은 React가 담당



```
const {barchartData, setBarchartData} = useBarchartState();  
const [data, setData] = useState<IBarchartData[]>();
```



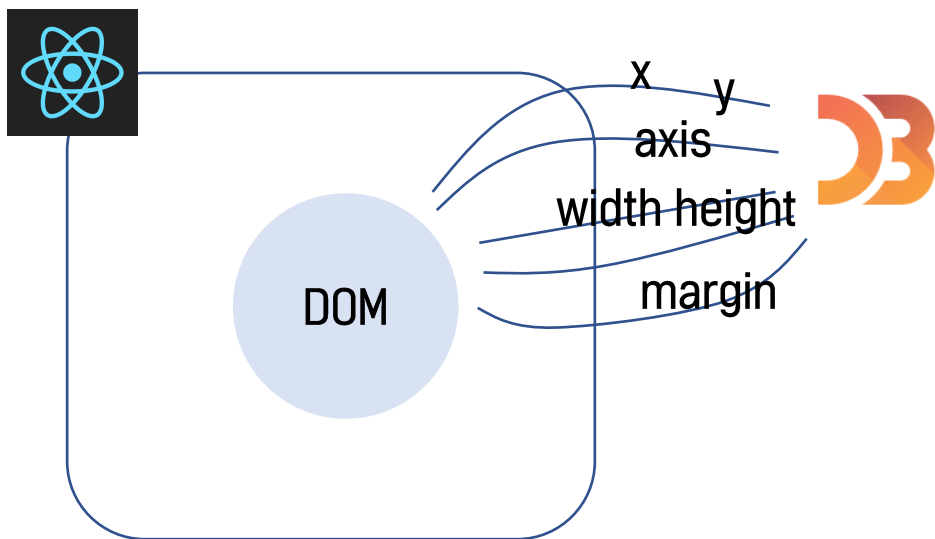
```
/* D3 좌표 계산 로직 작성 */  
const svgRef = useRef<any>();  
  
useEffect(() => {  
  setData(barchartData);  
  console.log(barchartData);  
}, [barchartData]);
```

# React와 D3의 결합성

> 접근 방법

3가지 접근 방법

## 3. 수학적 계산은 D3가, 렌더링은 React가 담당



```
useEffect(() => {  
  if(!data) return;  
  const margin = { top: 30, right: 30, bottom: 70, left: 40 },  
    width = 460 - margin.left - margin.right,  
    height = 400 - margin.top - margin.bottom;  
  
  // X axis  
  const svg = d3.select('svg')  
    .append("svg")  
    .attr("width", width + margin.left + margin.right)  
  
  ...  
  
  svg.selectAll("mybar")  
    .data(data)  
    .join("rect")  
    .attr("x", d => x(d.Country))  
    .attr("y", d => y(+d.Value))  
    .attr("width", x.bandwidth())  
    .attr("height", d => height - y(+d.Value))  
    .attr("fill", "#69b3a2")  
}, [data]);
```

```
return (  
  <div>  
    Bar chart..  
    { /* {barchartData.map((d, i) => {  
      return <div key={i}>{d.Country} {d.Value}</div>  
    })} */}  
    { /* 실제 DOM 조작 by React(또는 Next) */}  
    <svg width="500" height="500" className={styles.barchart}>  
      </svg>  
    </div>  
)
```



---

Angular는 상상 이상으로 D3와 궁합이 좋은 편이다!