Virtual DOM vs Real DOM

React

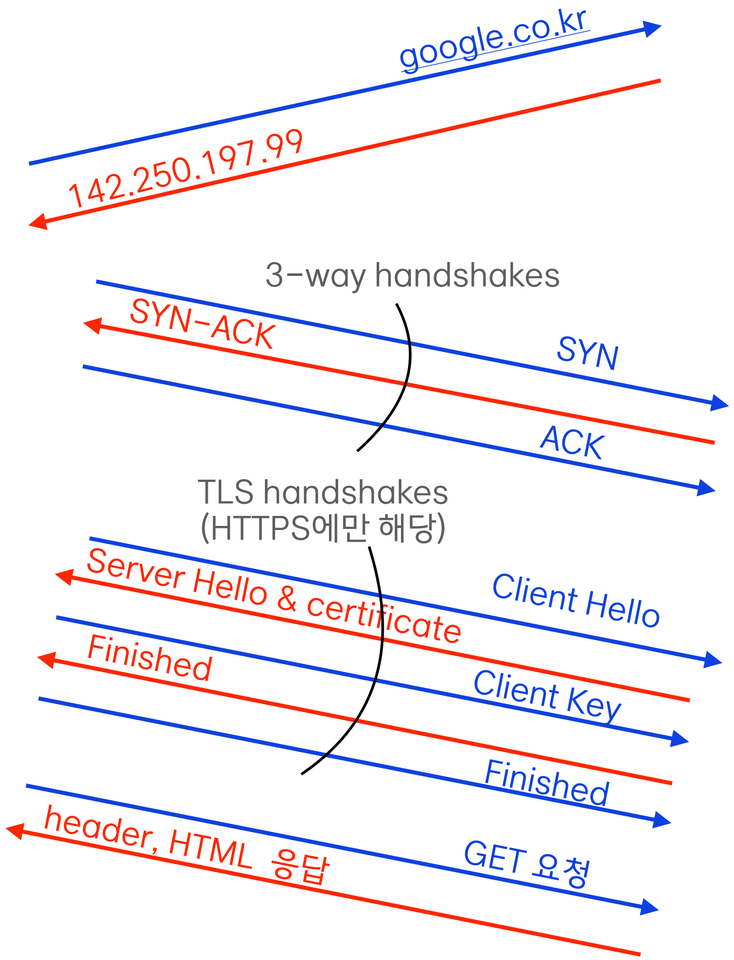
https://github.com/jio-ping

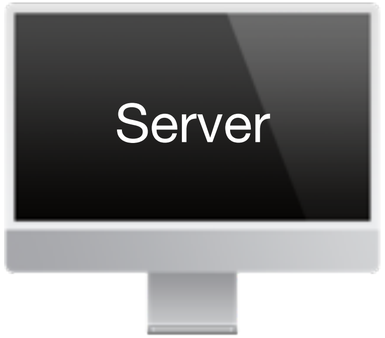
📌 브라우저 렌더링 과정 -1

주소창 입력부터 화면 렌더링까지

사용자가 주소창에
URL을 입력하게 되면,

DNS에서 URL의 도메인 이름을
IP 주소로 변환

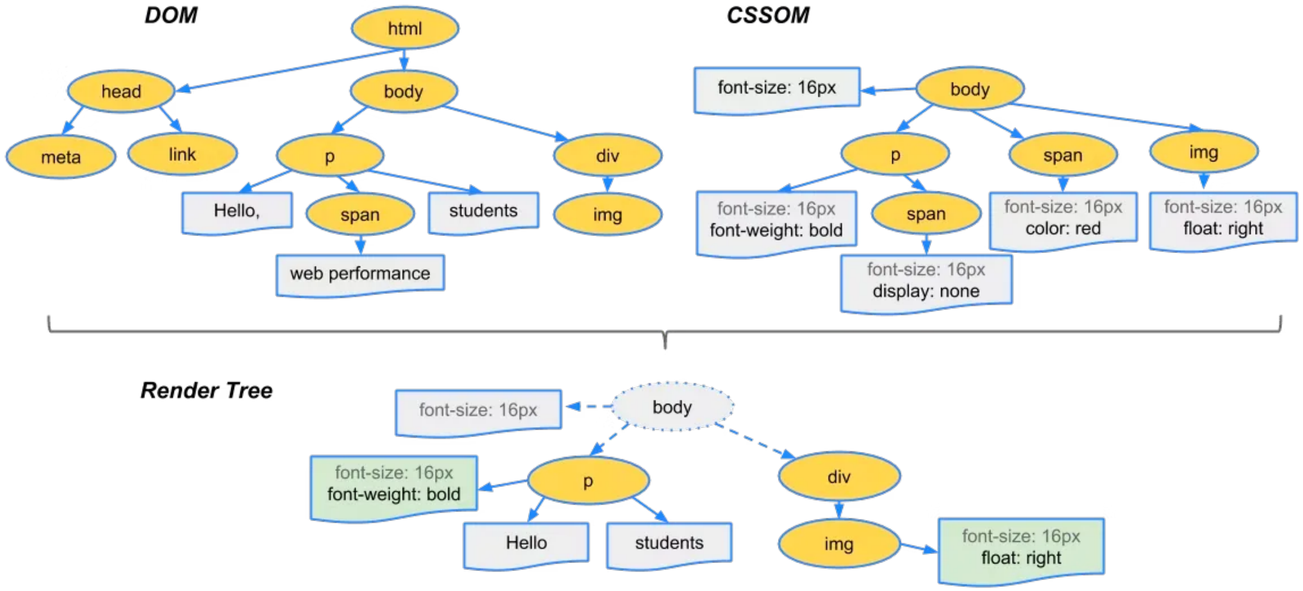




https://github.com/jio-ping

📌 브라우저 렌더링 과정 -2

주소창 입력부터 화면 렌더링까지



HTML과 CSS를 파싱해 DOM과 CSSOM이 생성되고 이를 합쳐서 Render Tree가 완성됨

Layout Tree

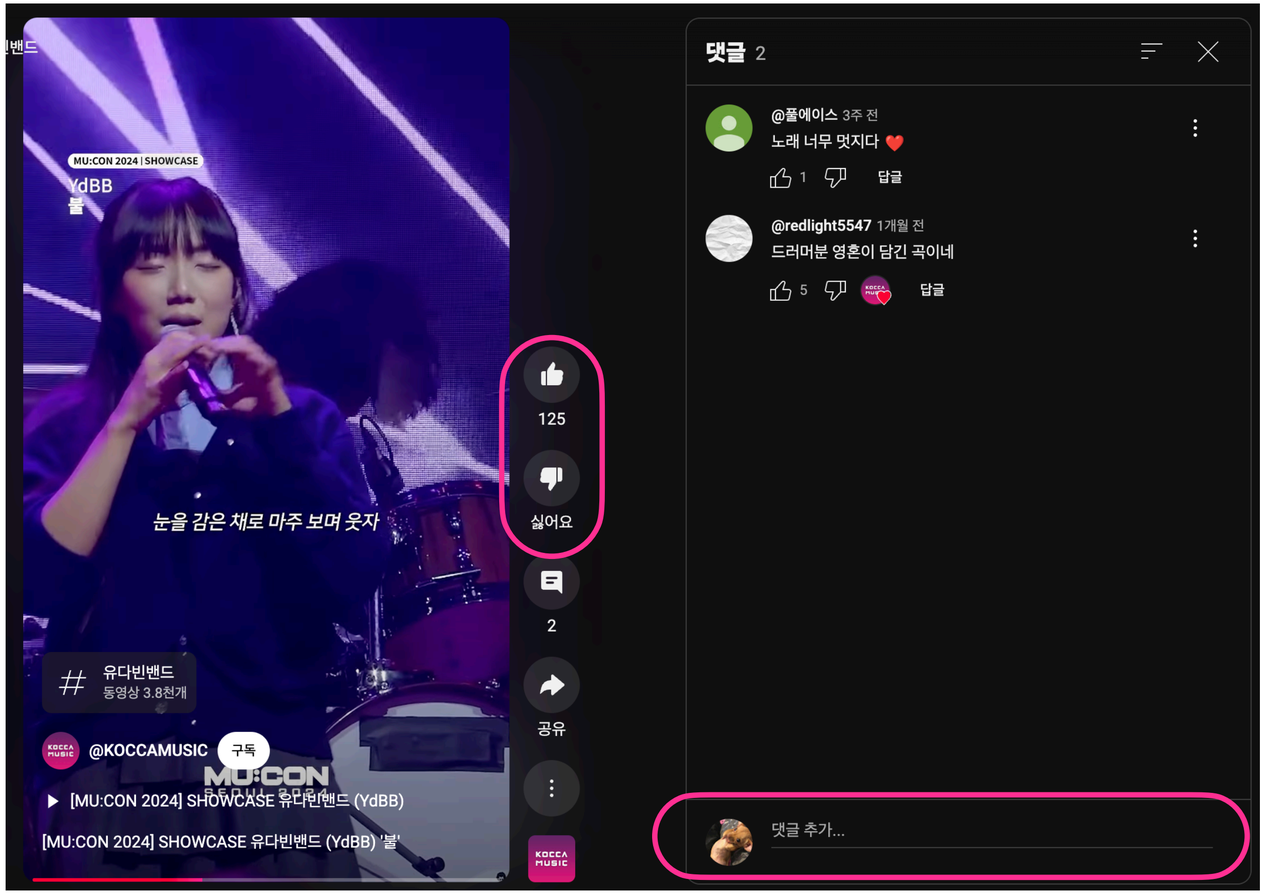
Layer Tree

Reflow
: DOM 요소가 추가되거나 제거될 때
크기, 위치가 변경될 때
반응형 레이아웃 조정 , 폰트크기변경
(성능적으로 비용이 매우 큰 작업)

Repaint
: 요소의 색상, 글꼴 스타일 변경
경계선 등 시각적 효과 변경

https://github.com/jio-ping

DOM을 조작해보자

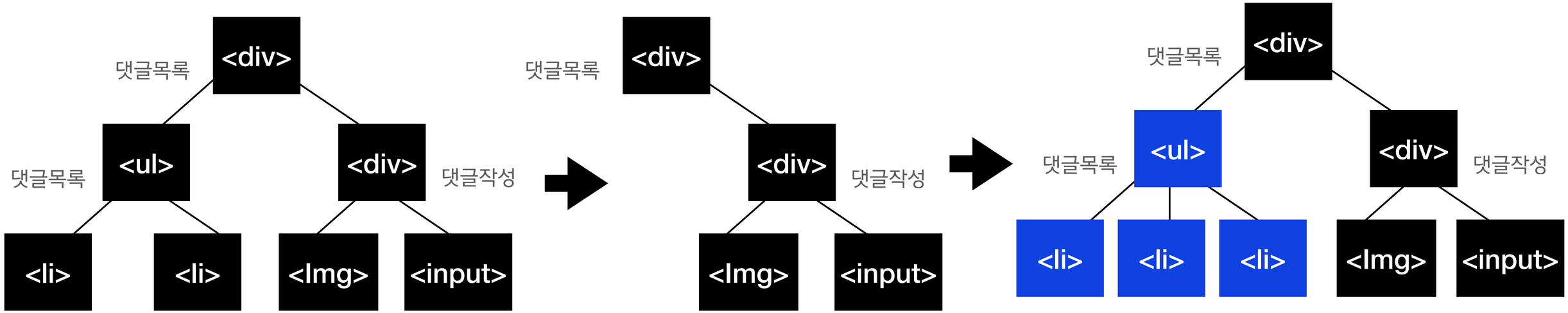


좋아요버튼을 누르거나 댓글을 추가하게 되면 DOM에선 무슨 일이 일어날까?

https://github.com/jio-ping

DOM을 조작해보자

댓글을 새로 작성했을때 생기는 일



https://github.com/jio-ping

Real DOM의 특징과 한계

React와 vue.js, solid는 왜 virtual DOM을 적용했을까?

Real DOM

렌더링

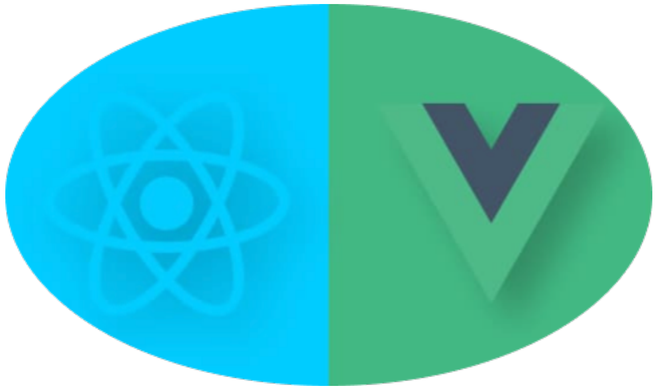
HTML 파싱해 DOM 트리 생성 후 화면에 표시

업데이트

DOM을 직접 수정하면, 브라우저는 Reflow & Repaint 수행

1. DOM트리의 변경사항 처리 **시 하위 노드까지 업데이트**
2. 변경사항이 생길때마다 **Repaint & Reflow 발생 가능성**
3. 어떻게 변경할지 명령형 방식으로 제어해 조작 순서와 상태 동기화가 까다로움

Virtual DOM



변경해야 될 노드만 변경하고
여러 변경사항을 한 번에 처리하고
무엇을 보여줄 지만 선언하면 어떨까?

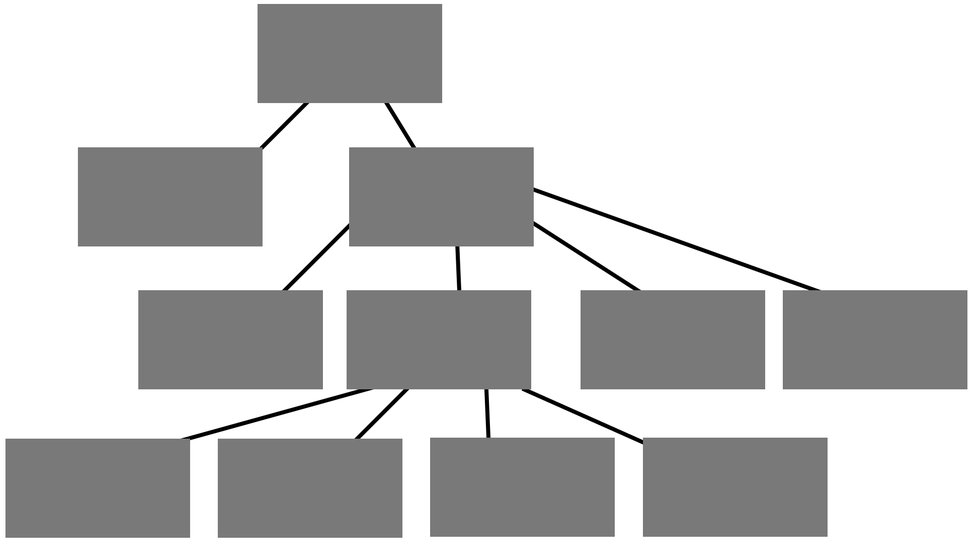
https://github.com/jio-ping

React의 Virtual DOM

: UI의 이상적인 또는 “가상”적인 표현을 메모리에 저장하고 ReactDOM과 같은 라이브러리에 의해 “실제” DOM과 동기화하는 프로그래밍 개념

Reconciliation

Initial Rendering 이후 state나 props가 변경되면 Re-rendering이 trigger 돼 **새로운 virtual DOM이 생성됨**



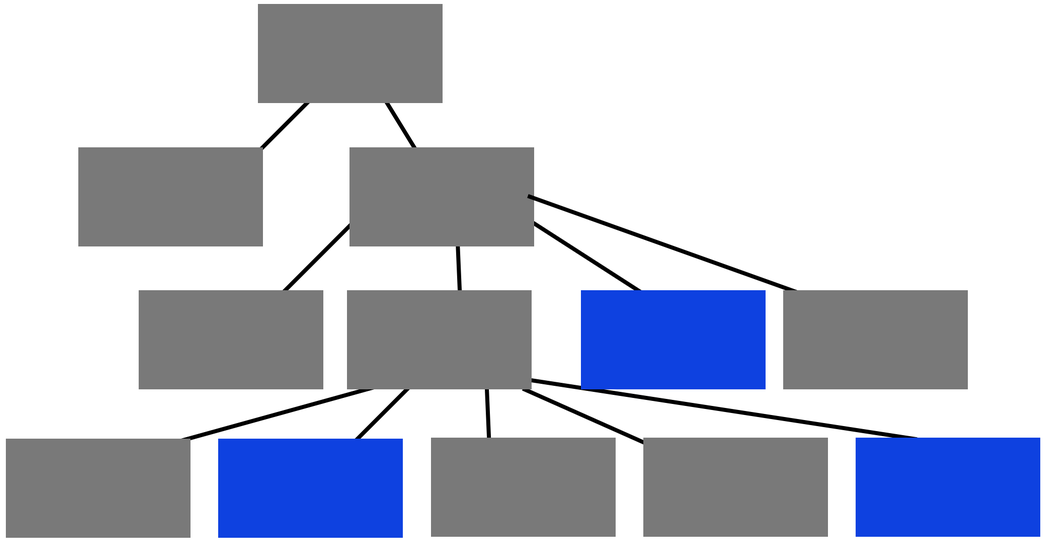
Previous Virtual DOM

: (현재) 변경 이전의 내용

Diffing

이전버전과 virtual DOM을
비교해 변경될 부분을 감지

한번에 (batch Updates)
실제 DOM에 반영



Virtual DOM

: 변경 이후에 보여질 내용

https://github.com/jio-ping

Virtual DOM

어째서 효율적이라는 걸까?

Batch Updates

여러 업데이트를 하나의 작업(batch)으로 처리하여 성능을 최적화함 ➡ 리액트의 순수성 유지와도 관련

```jsx
function App() {
  const [count1, setCount1] = useState(0);
  const [count2, setCount2] = useState(0);

  const handleClick = () => {
    setCount1((prev) => prev + 1); // state 변경
    setCount2((prev) => prev + 1); // state 변경
  };

  return <button onClick={handleClick}>Click
Me</button>;
}
```

state를 두 번 변경해도 (=두 개의 업데이트) 하나의 배치로 처리

State 업데이트 요청

: React는 여러 state 업데이트 요청을 업데이트 큐에 추가.

↓

Reconciliation

: 모든 state 변경을 반영 후 diffing algorithm을 통해 비교

↓

Commit Phase

: 최종적으로 Virtual DOM 변경 사항을 DOM에 반영

https://github.com/jio-ping

변경을 감지하는 방법

Diffing Algorithm은 어떻게 작동할까?😎

엘리먼트의 타입이 같거나/다른 경우

```
<div>
  <Counter />
</div>

<span>
  <Counter />
</span>
```

두 루트 엘리면트 타입이 다를땐
이전 트리를 버리고 완전히 새로운 트리를 구축
하위 컴포넌트는 언마운트되고 state도 사라짐
-> Conter 사라지고, 새로 다시 마운트

```
<div className="before" title="stuff"
/>
<div className="after" title="stuff" />
```

두 엘리먼트의 속성을 확인하여, 동일한 내역은 유지하고 변경된 속성들만 갱신
-> className만 변경

자식요소에 대한 재귀적 처리

```
<ul>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>

<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

동시에 두 리스트를 순회하고 차이점이 있으면 변경을 생성
Key 속성으로 일치여부를 판단함
Key 속성이 없을땐 마지막 순서에 추가가 되는 경우엔 크게 문제가 없겠지만
첫번째가 추가되면 ul요소의 모든 자식 요소를 다시 그림
(index를 key속성으로 지정하는 걸 지양해야되는 이유)

서로 다른 타입의 두 엘리먼트는 서로 다른 트리를 만든다.
개발자가 key prop을 통해, 여러 렌더링 사이에서 어떤 자식 엘리먼트가 변경되지 않아야 할지 표시해 준다.

https://github.com/jio-ping

Reference

브라우저 렌더링 원리
https://developer.mozilla.org/ko/docs/Web/Performance/How_browsers_work

Virtual DOM
https://ko.legacy.reactjs.org/docs/faq-internals.html

Reconciliation - Diff 알고리즘
https://ko.legacy.reactjs.org/docs/reconciliation.html