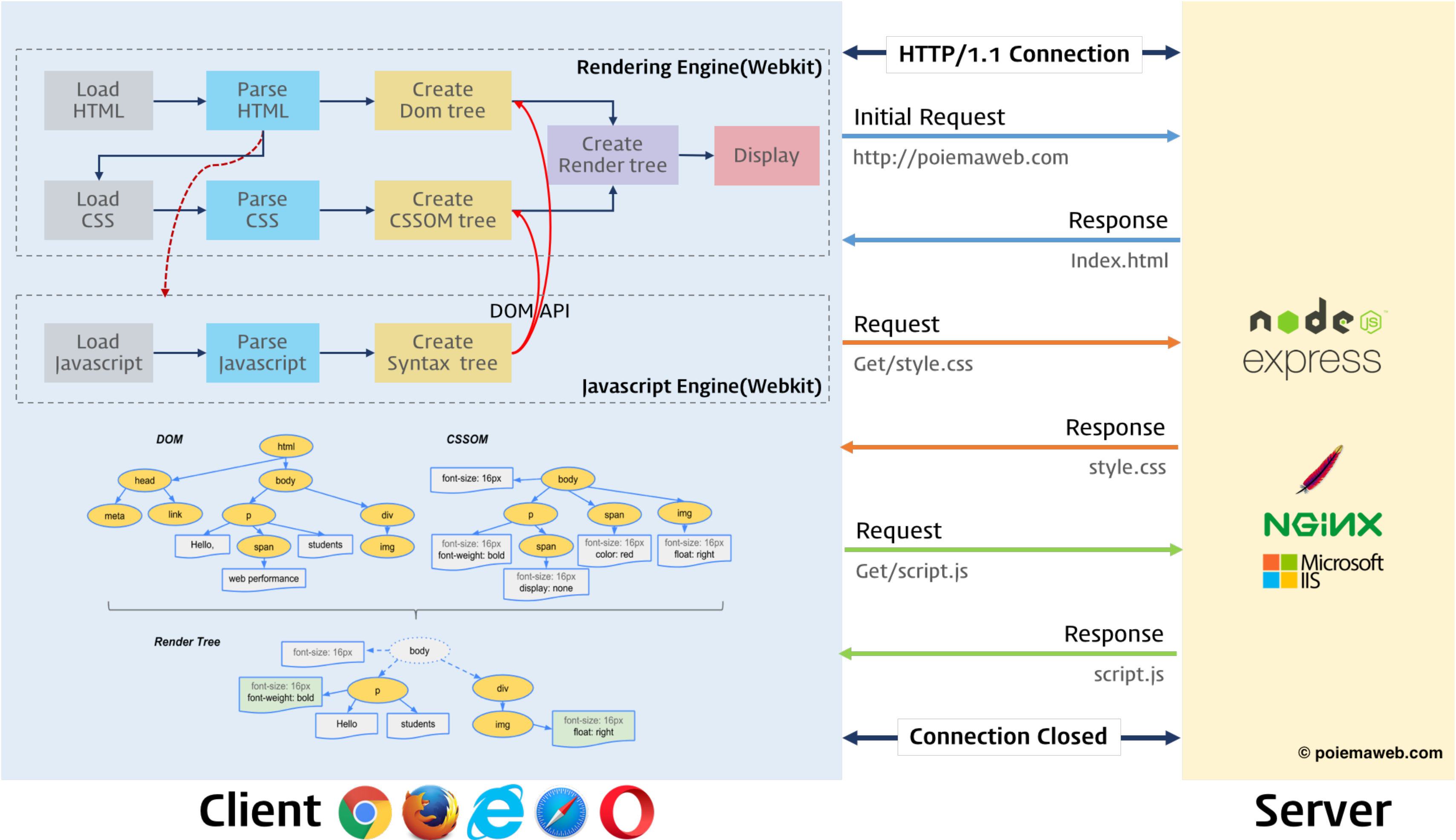


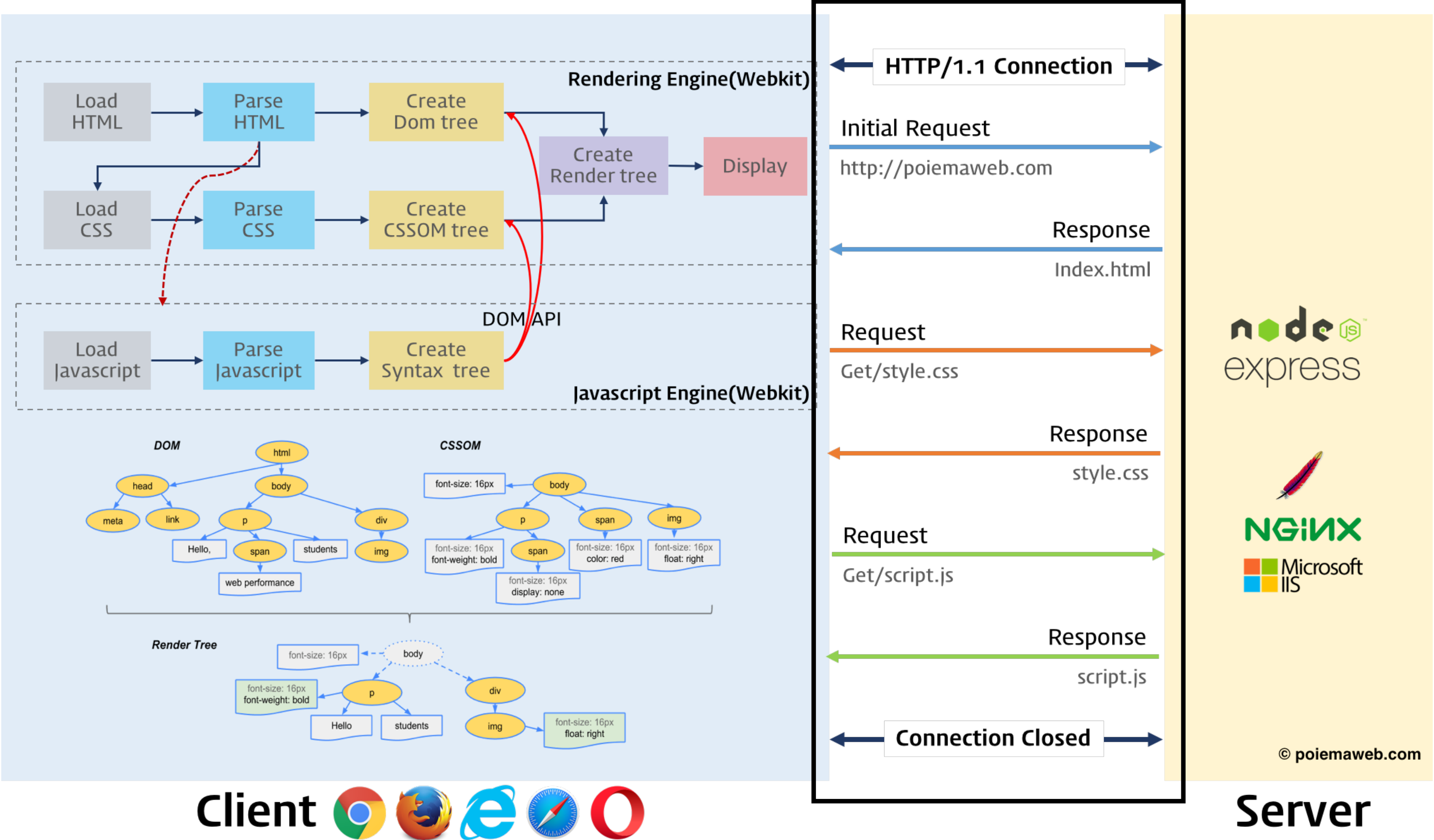
38. 브라우저의 렌더링 과정

2022.04.02

브라우저의 간략한 렌더링 과정

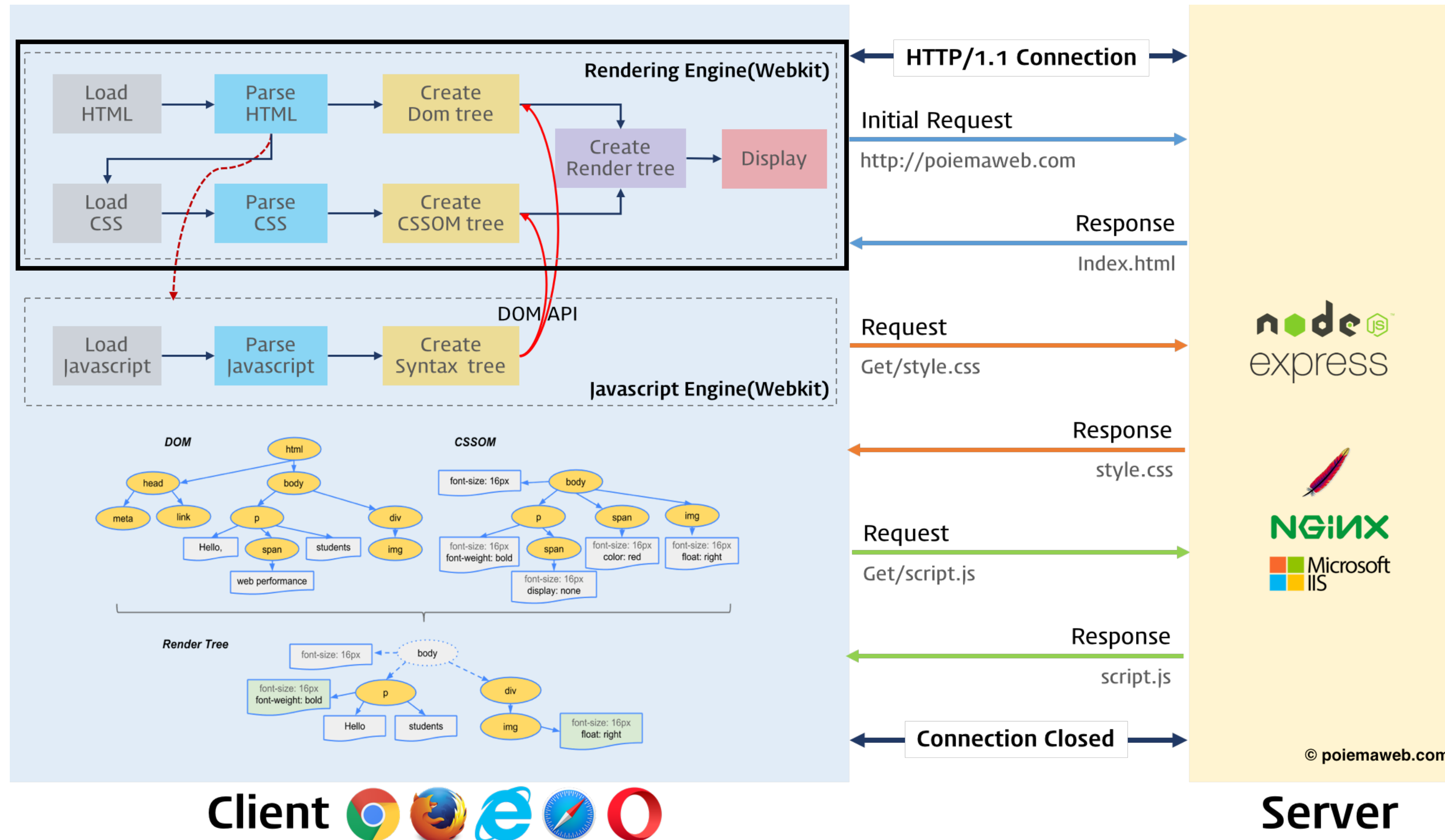


브라우저의 간략한 렌더링 과정



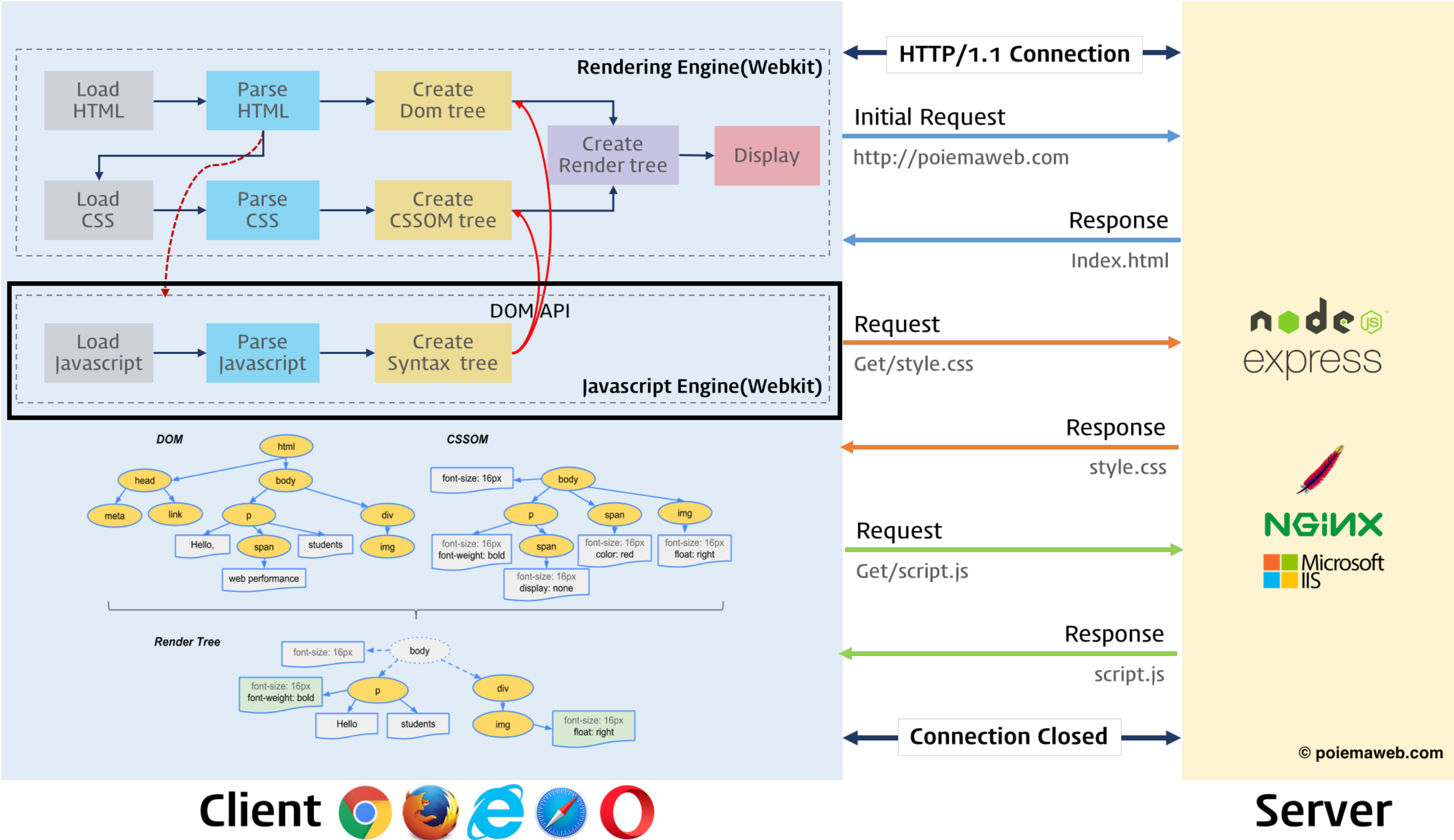
1. 브라우저는 HTML, CSS, 자바스크립트, 이미지, 폰트 파일 등 렌더링에 필요한 리소스를 요청하고 서버로부터 응답을 받는다.

브라우저의 간략한 렌더링 과정



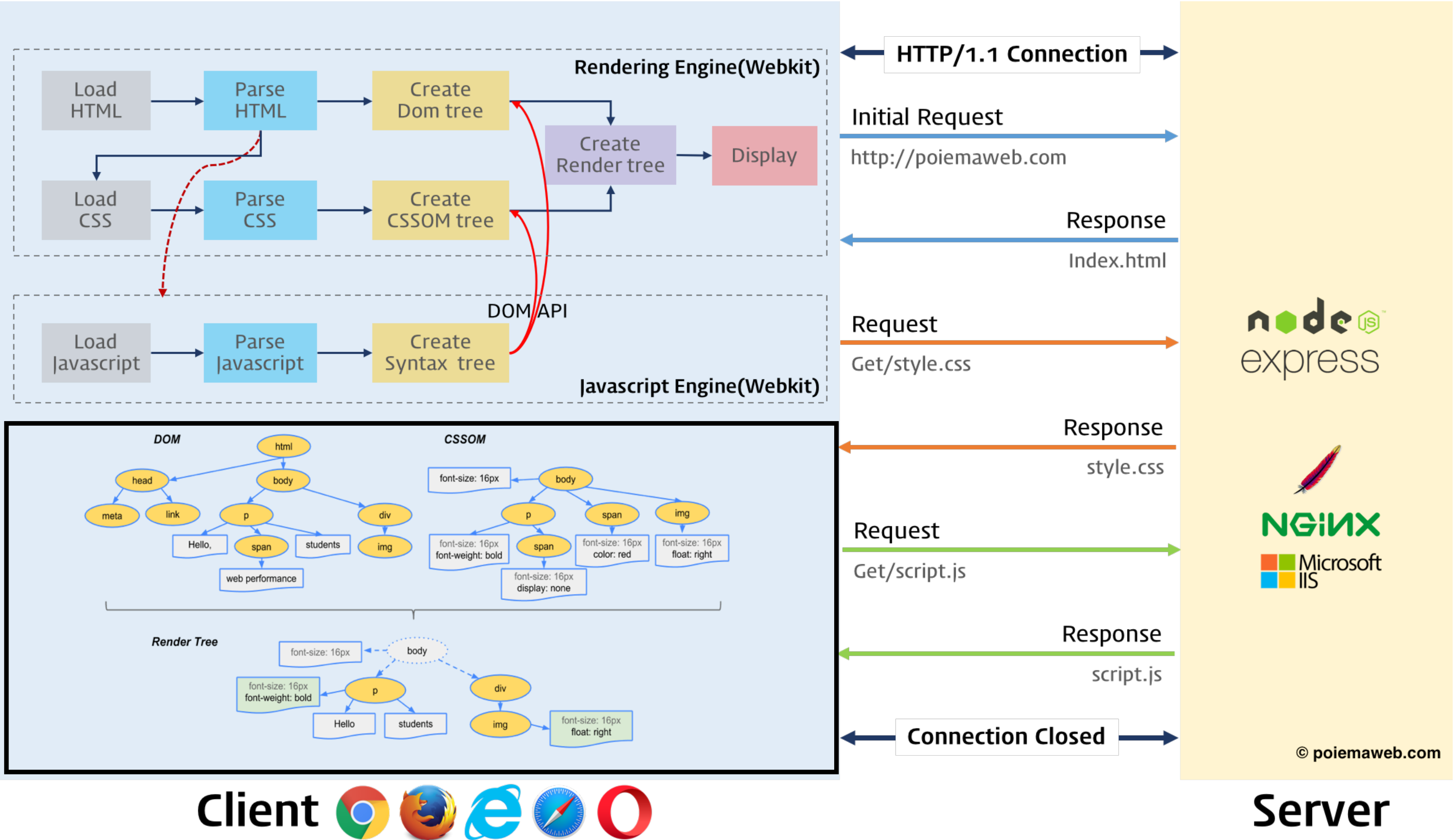
2. 서버로부터 응답된 HTML과 CSS를 파싱하여 DOM과 CSSOM을 생성하고 이들을 결합하여 렌더 트리를 생성한다.

브라우저의 간략한 렌더링 과정



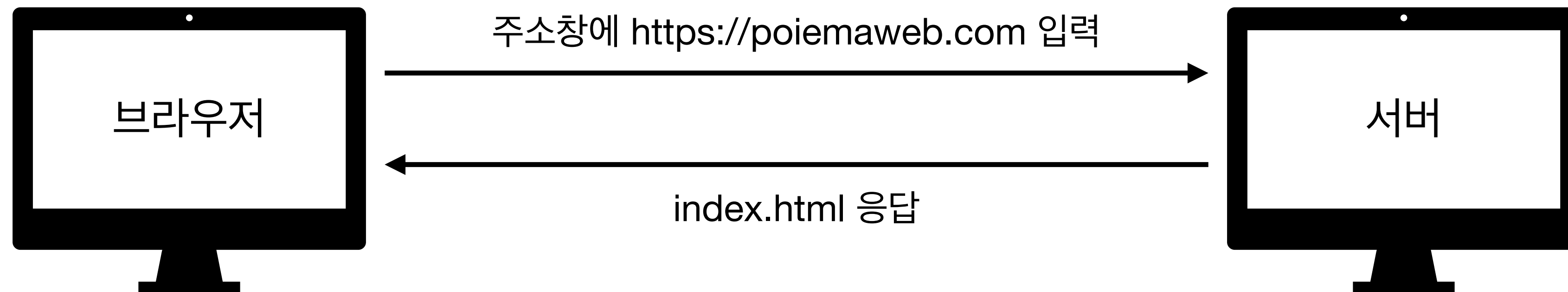
3. 서버로부터 응답된 자바스크립트를 파싱하여 AST를 생성하고 바이트코드로 변환하여 실행한다.
이 때 DOM API를 통해 DOM이나 CSSOM을 변경할 수 있다. 변경된 DOM과 CSSOM은 다시 렌더 트리으로 결합된다.

브라우저의 간략한 렌더링 과정



4. 렌더 트리를 기반으로 HTML 요소의 레이아웃을 계산하고 브라우저 화면에 HTML 요소를 페인팅한다.

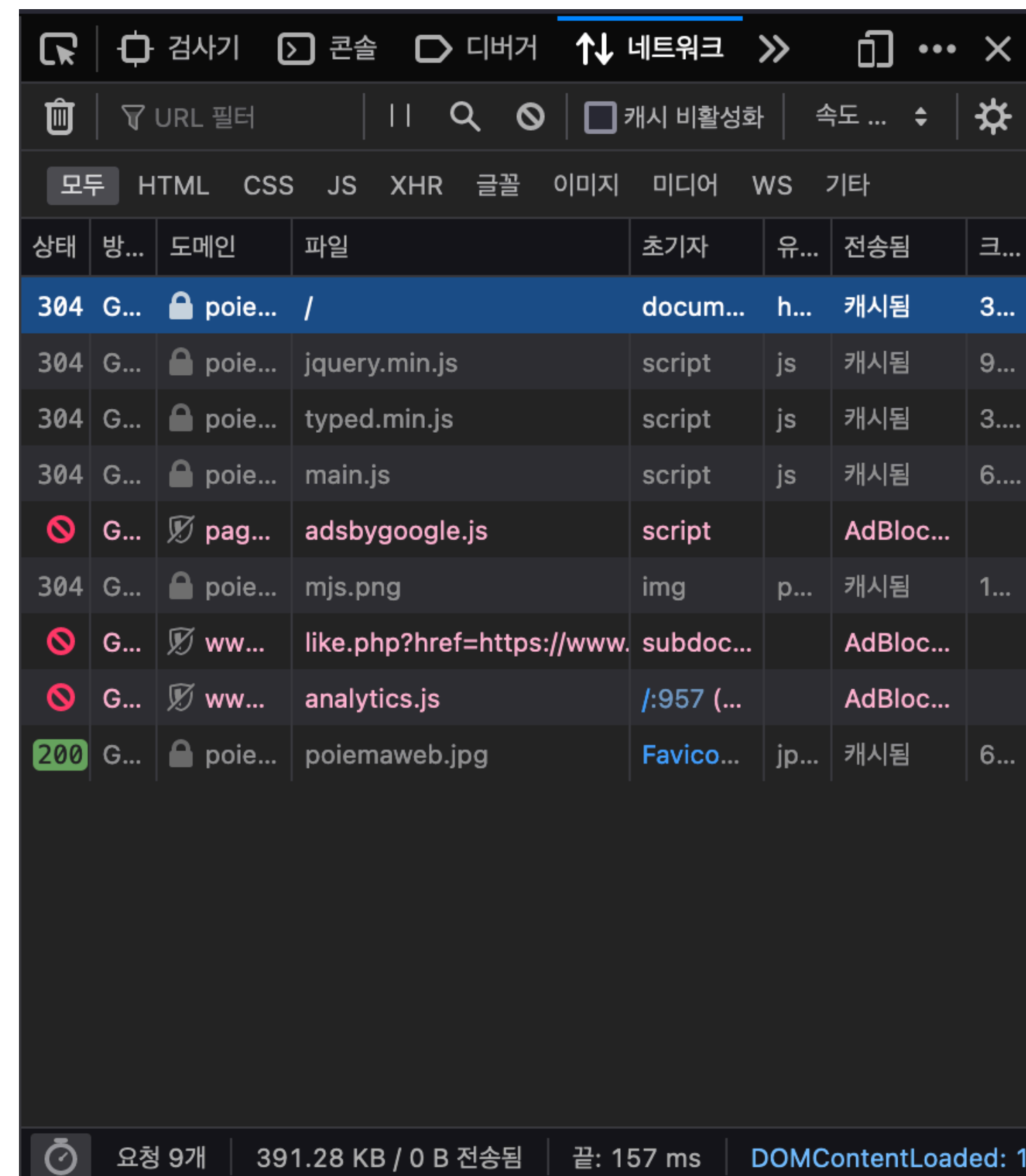
1. 요청과 응답



1. 요청과 응답

서버에 요청한 내용과 서버가 응답한 내용은 개발자 도구의 네트워크 패널에서 확인할 수 있다.

HTML을 파싱하는 도중에 요청되는 리소스 파일들



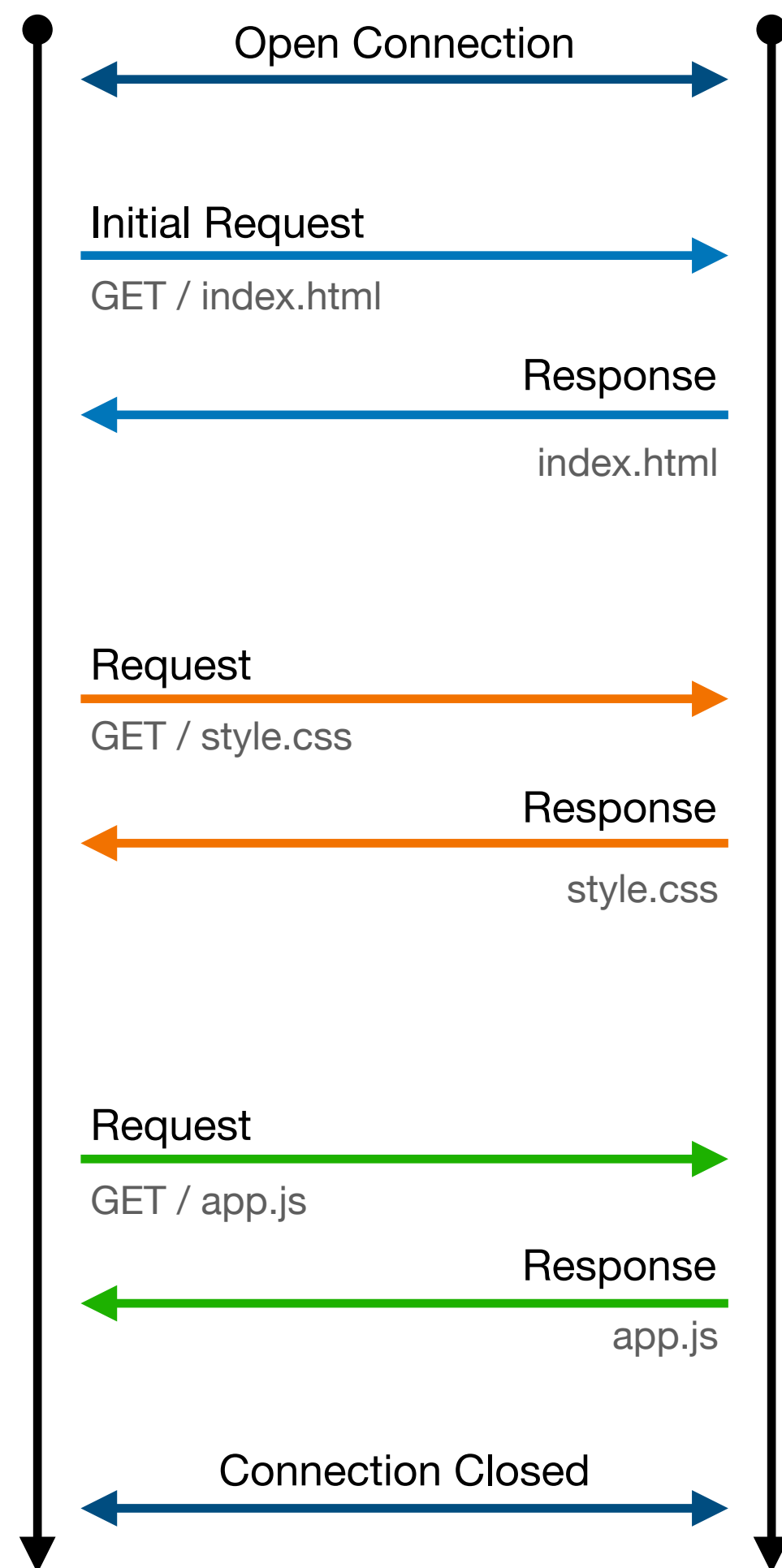
상태	방...	도메인	파일	초기자	유...	전송됨	크...
304	G...	poie...	/	docum...	h...	캐시됨	3...
304	G...	poie...	jquery.min.js	script	js	캐시됨	9...
304	G...	poie...	typed.min.js	script	js	캐시됨	3...
304	G...	poie...	main.js	script	js	캐시됨	6...
🚫	G...	pag...	adsbygoogle.js	script		AdBloc...	
304	G...	poie...	mjs.png	img	p...	캐시됨	1...
🚫	G...	ww...	like.php?href=https://www.	subdoc...		AdBloc...	
🚫	G...	ww...	analytics.js	/:957 (...)		AdBloc...	
200	G...	poie...	poiemaweb.jpg	Favico...	jp...	캐시됨	6...

요청 9개 | 391.28 KB / 0 B 전송됨 | 끝: 157 ms | DOMContentLoaded: 1

index.html

2. HTTP 1.1과 HTTP 2.0

HTTP 1.1

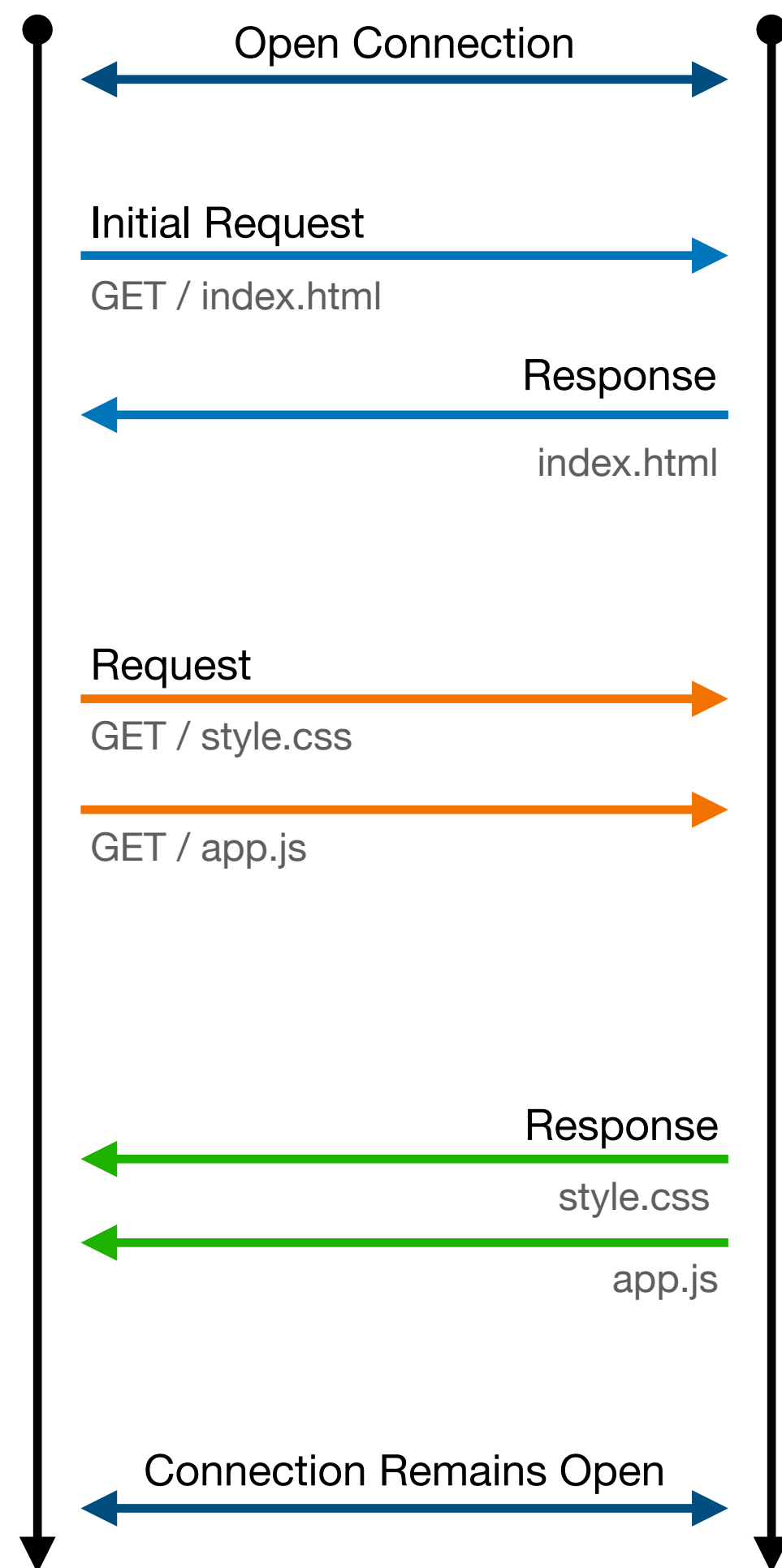


1. 커넥션당 하나의 요청과 응답만 처리한다.

2. 요청할 리소스의 개수에 비례하여 응답 시간도 증가한다는 단점이 있다.

2. HTTP 1.1과 HTTP 2.0

HTTP 2.0



1. 커넥션당 여러 개의 요청과 응답을 처리한다.

2. HTTP 1.1에 비해 페이지 로드 속도가 약 50% 정도 빠르다고 알려져 있다.

3. HTML 파싱과 DOM 생성

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

3. HTML 파싱과 DOM 생성

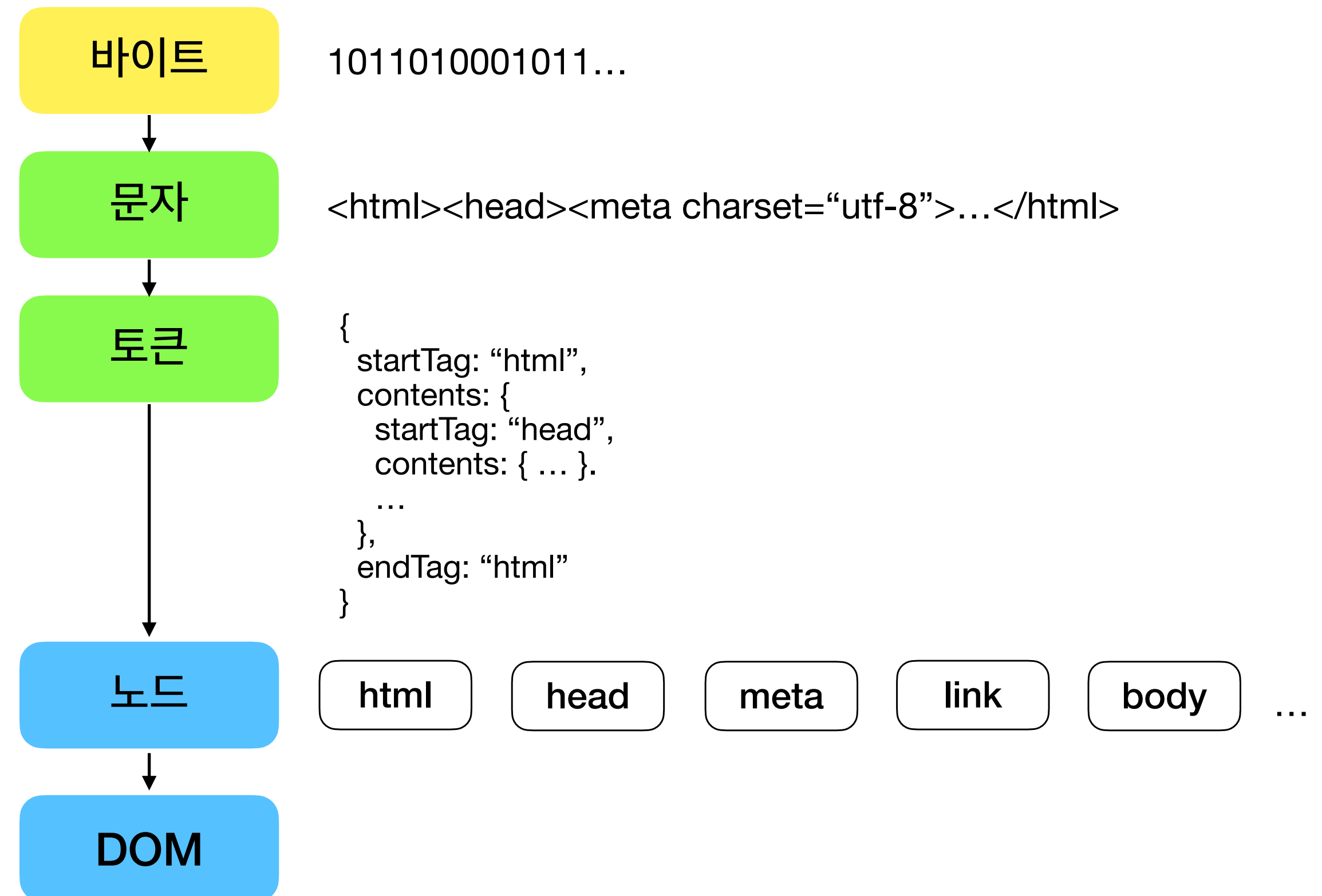
HTML은 순수한 텍스트이므로
브라우저가 이해할 수 있는 자료구조로 변환하여 메모리에 저장해야한다.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

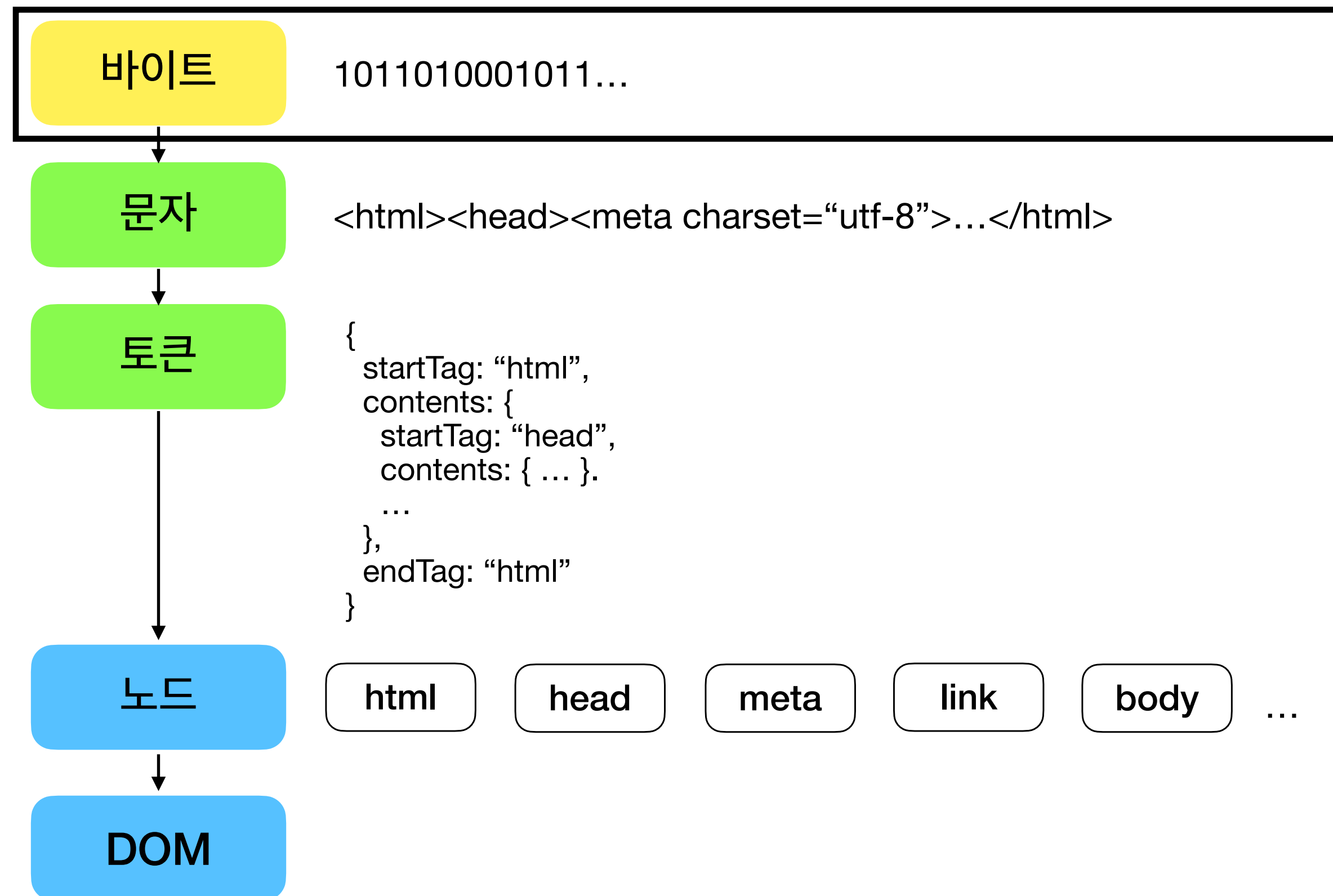
3. HTML 파싱과 DOM 생성

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

HTML은 순수한 텍스트이므로
브라우저가 이해할 수 있는 자료구조로 변환하여 메모리에 저장해야한다.

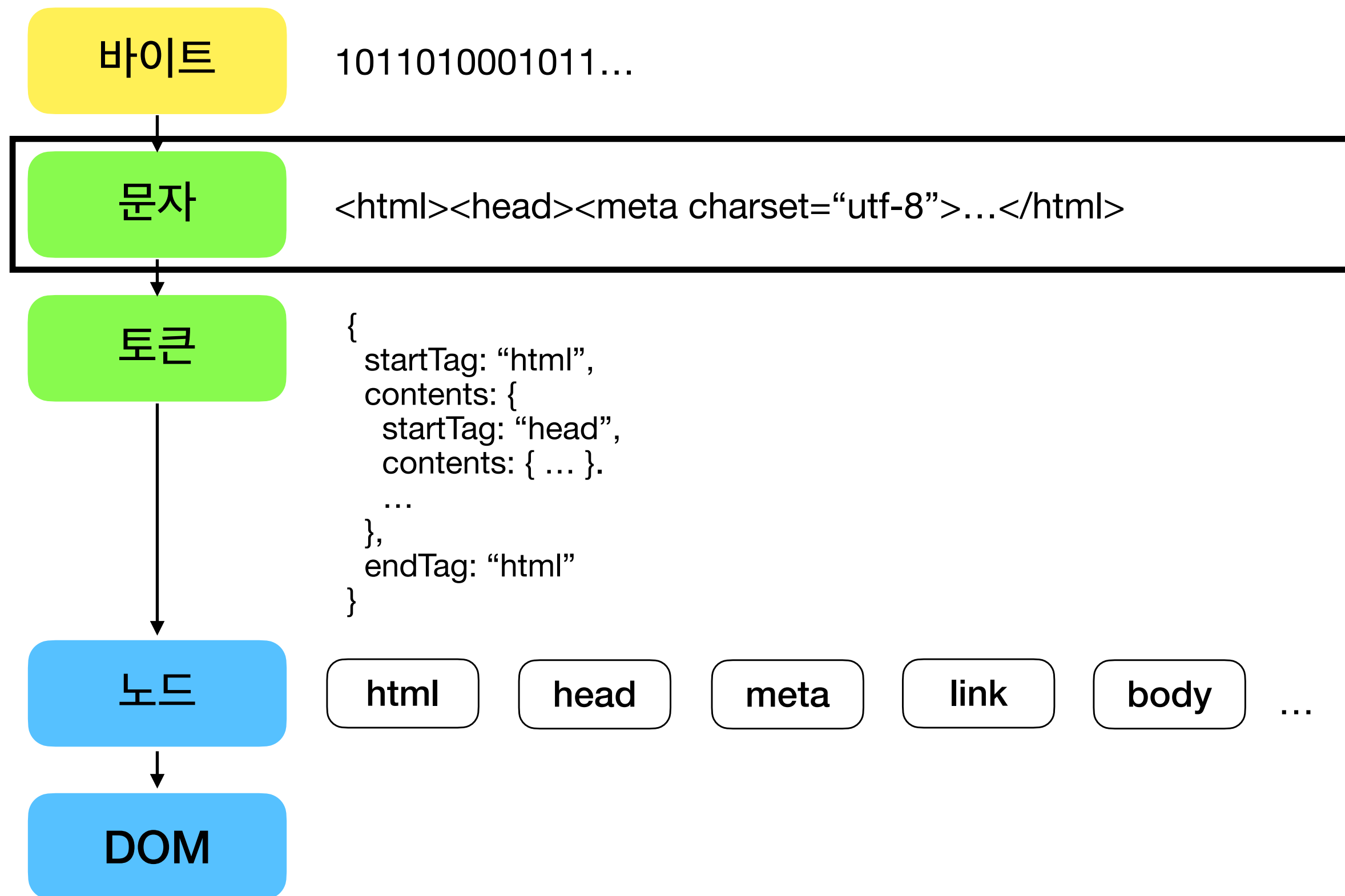


3. HTML 파싱과 DOM 생성



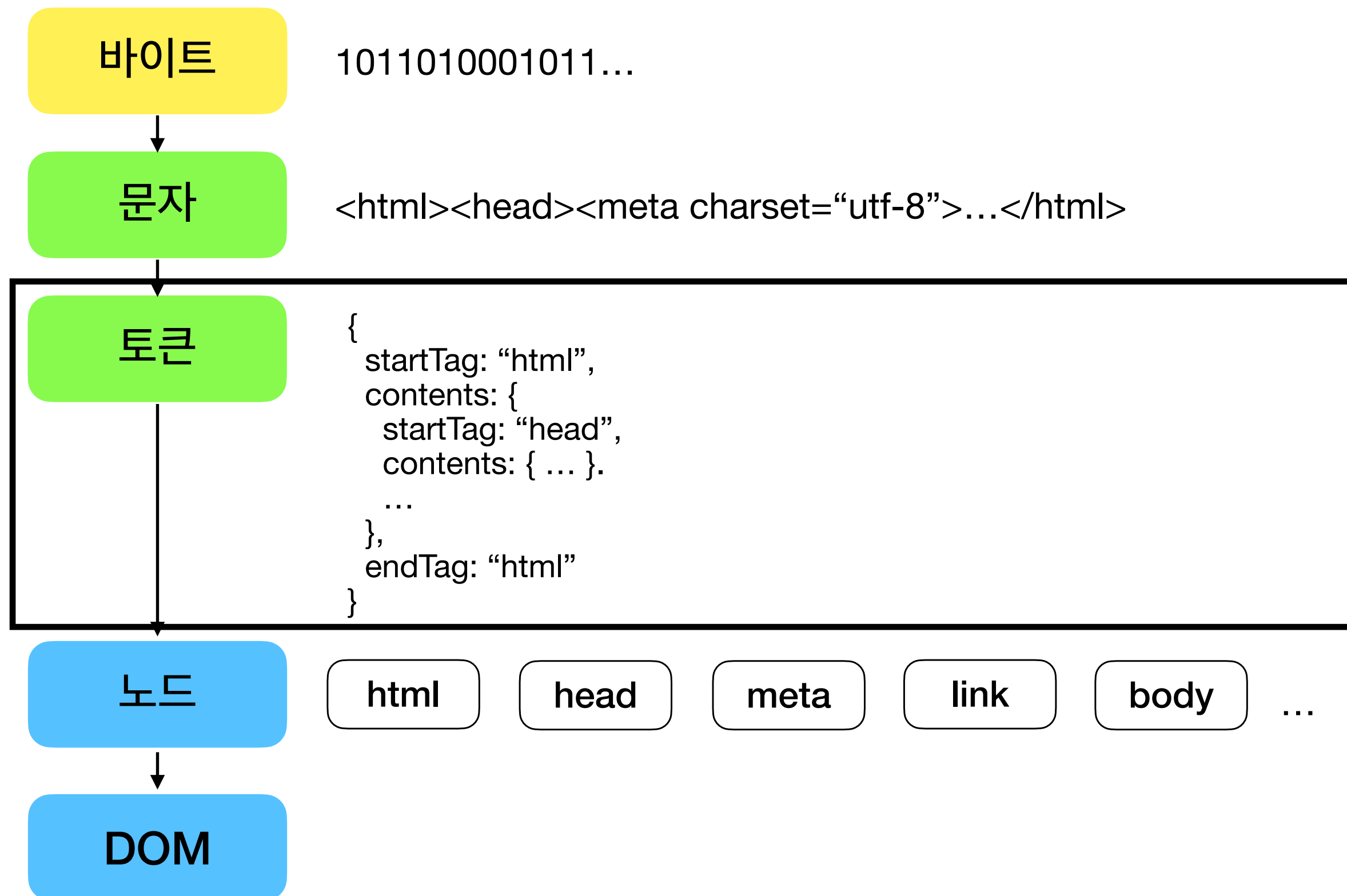
1. 서버에 존재하던 HTML 파일이 브라우저 요청에 의해 응답된다.

3. HTML 파싱과 DOM 생성



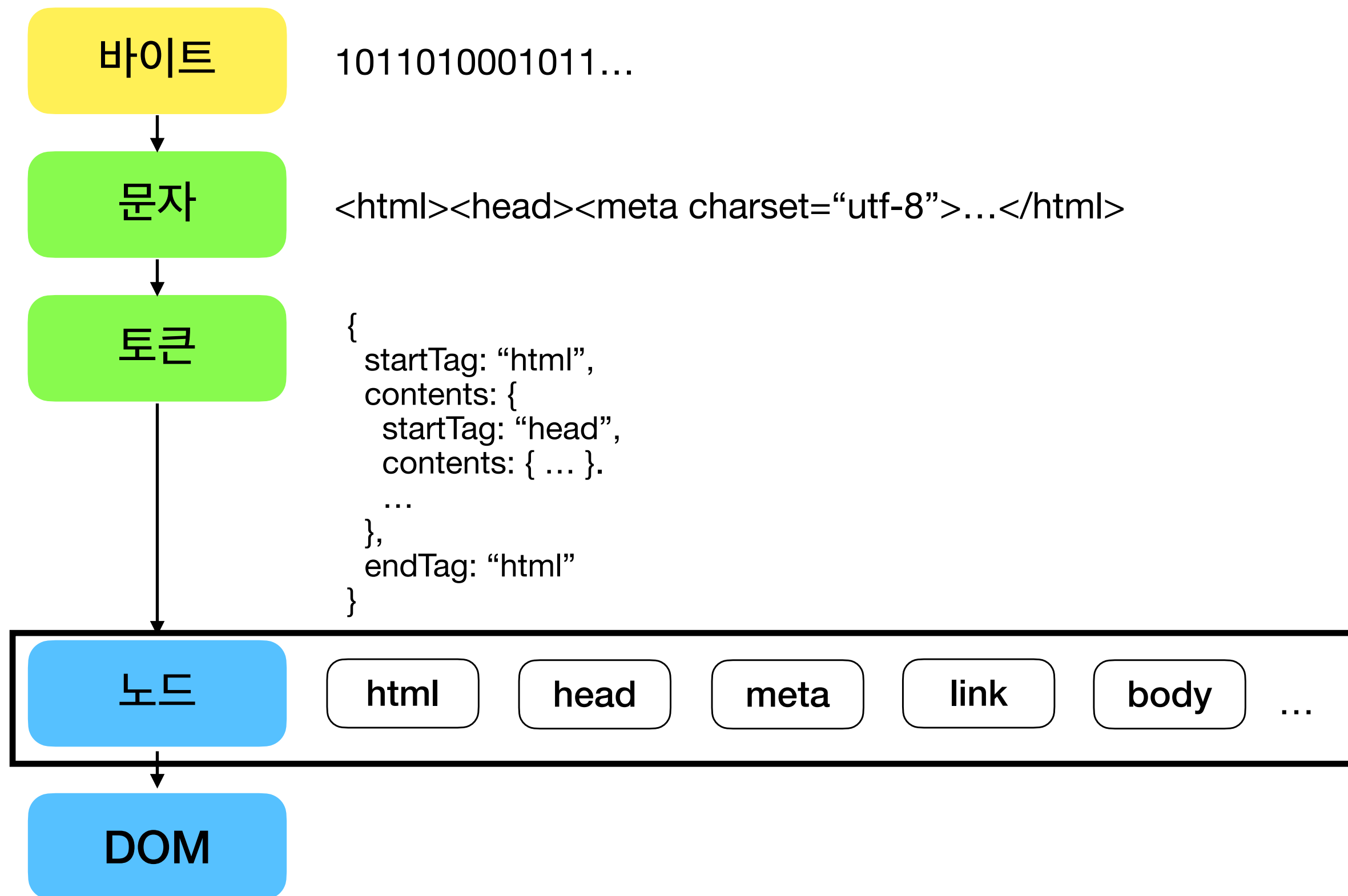
2. 바이트 형태로 응답받은 HTML 파일을 문자열로 변환한다.

3. HTML 파싱과 DOM 생성



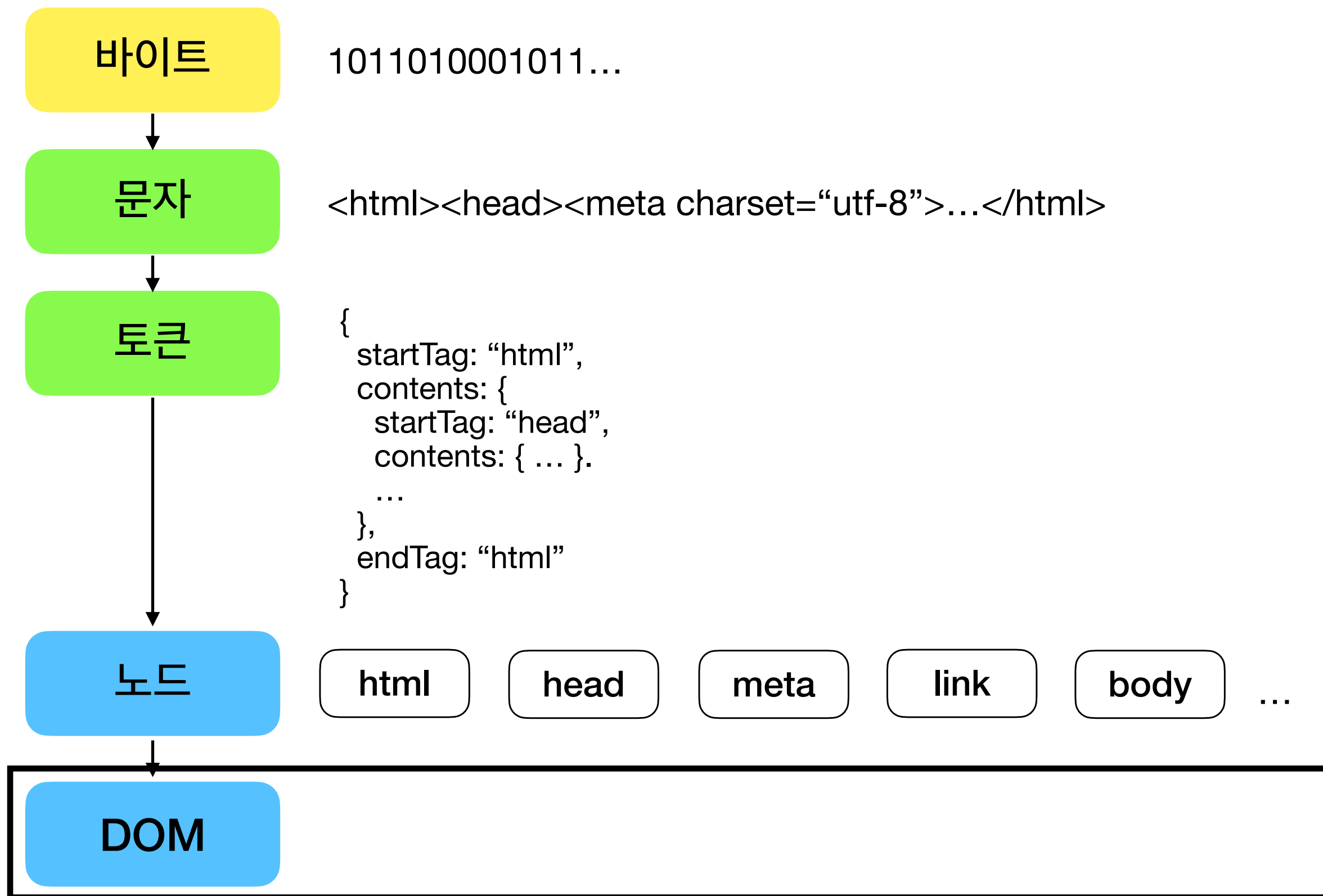
3. 문자열로 변환된 HTML 파일을 읽어 들여 문법적 의미를 갖는 코드의 최소 단위인 **토큰**들로 분해한다.

3. HTML 파싱과 DOM 생성



4. 각 토큰들을 객체로 변환하여 노드를 생성한다.

3. HTML 파싱과 DOM 생성

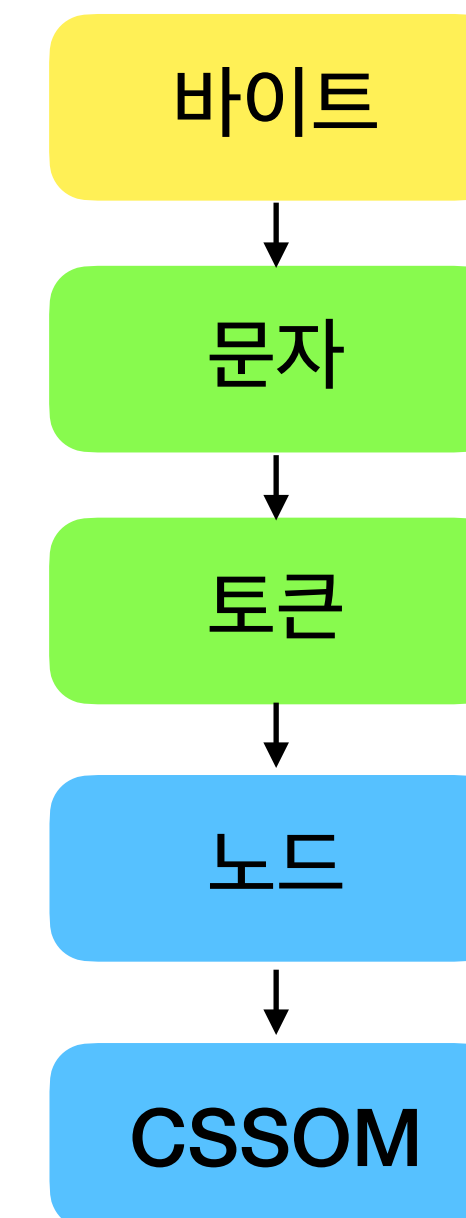


5. 모든 노드들을 트리 자료구조로 구성한다. (DOM 생성)

4. CSS 파싱과 CSSOM 생성

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

CSS도 HTML과 동일한 파싱과정을 거치며 CSSOM이 생성된다.



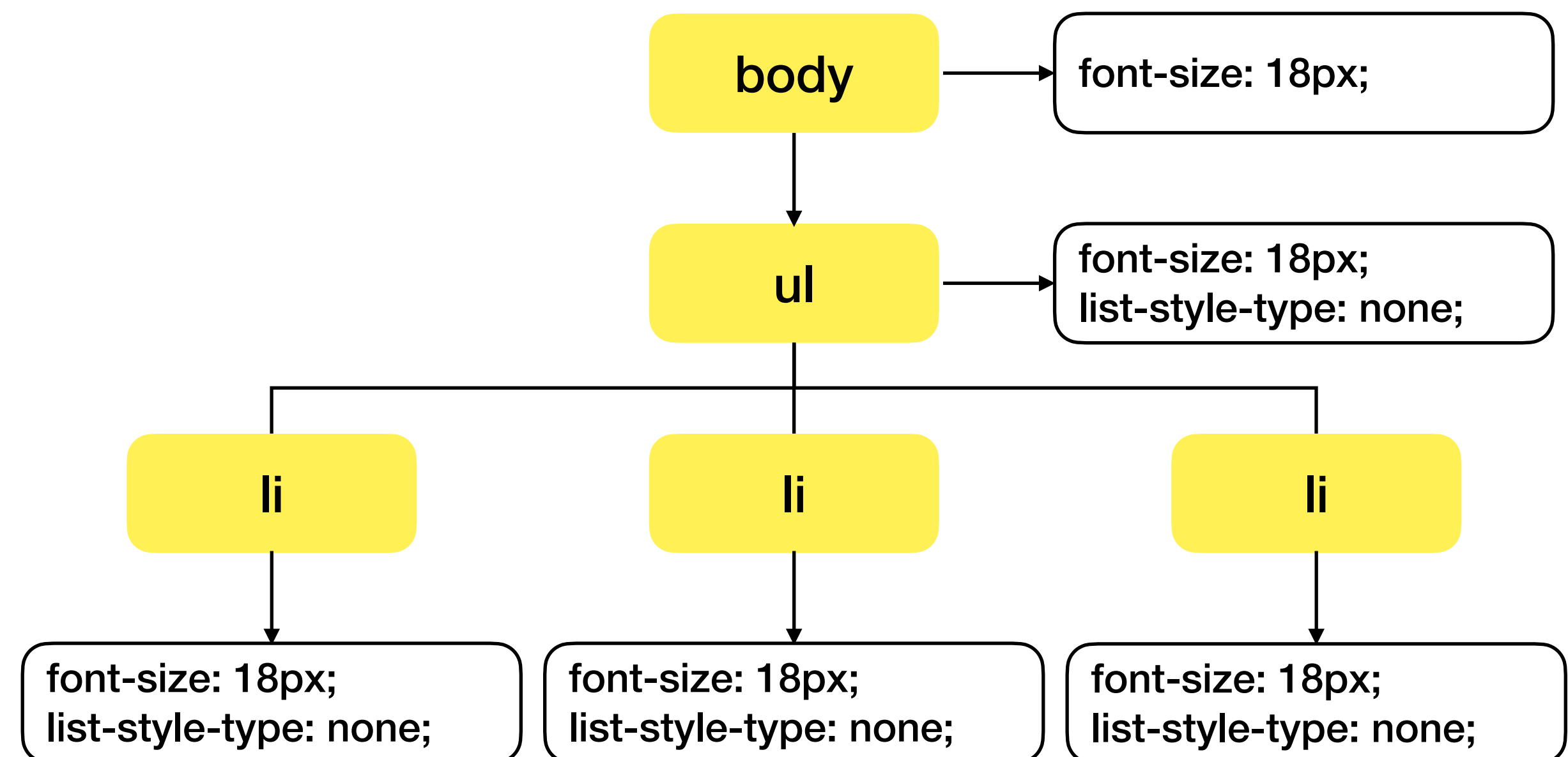
4. CSS 파싱과 CSSOM 생성



```
body {  
  font-size: 18px;  
}  
  
ul {  
  list-style-type: none;  
}
```

4. CSS 파싱과 CSSOM 생성

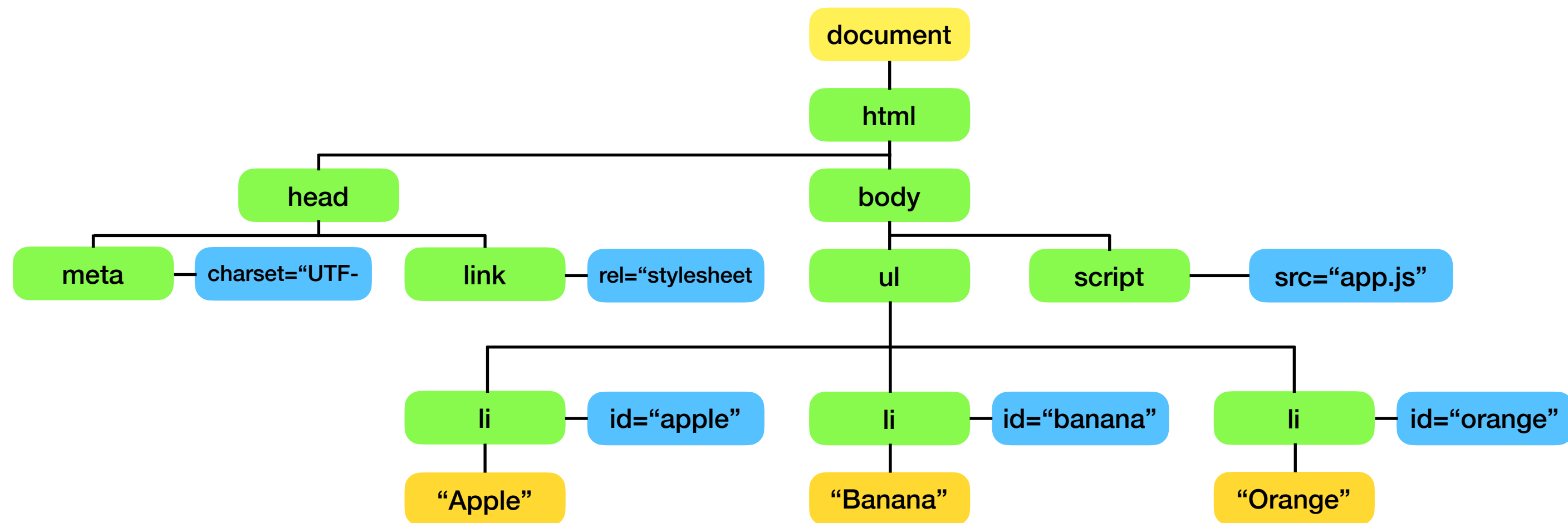
```
body {  
  font-size: 18px;  
}  
  
ul {  
  list-style-type: none;  
}
```



CSSOM은 CSS의 상속을 반영하여 생성된다.

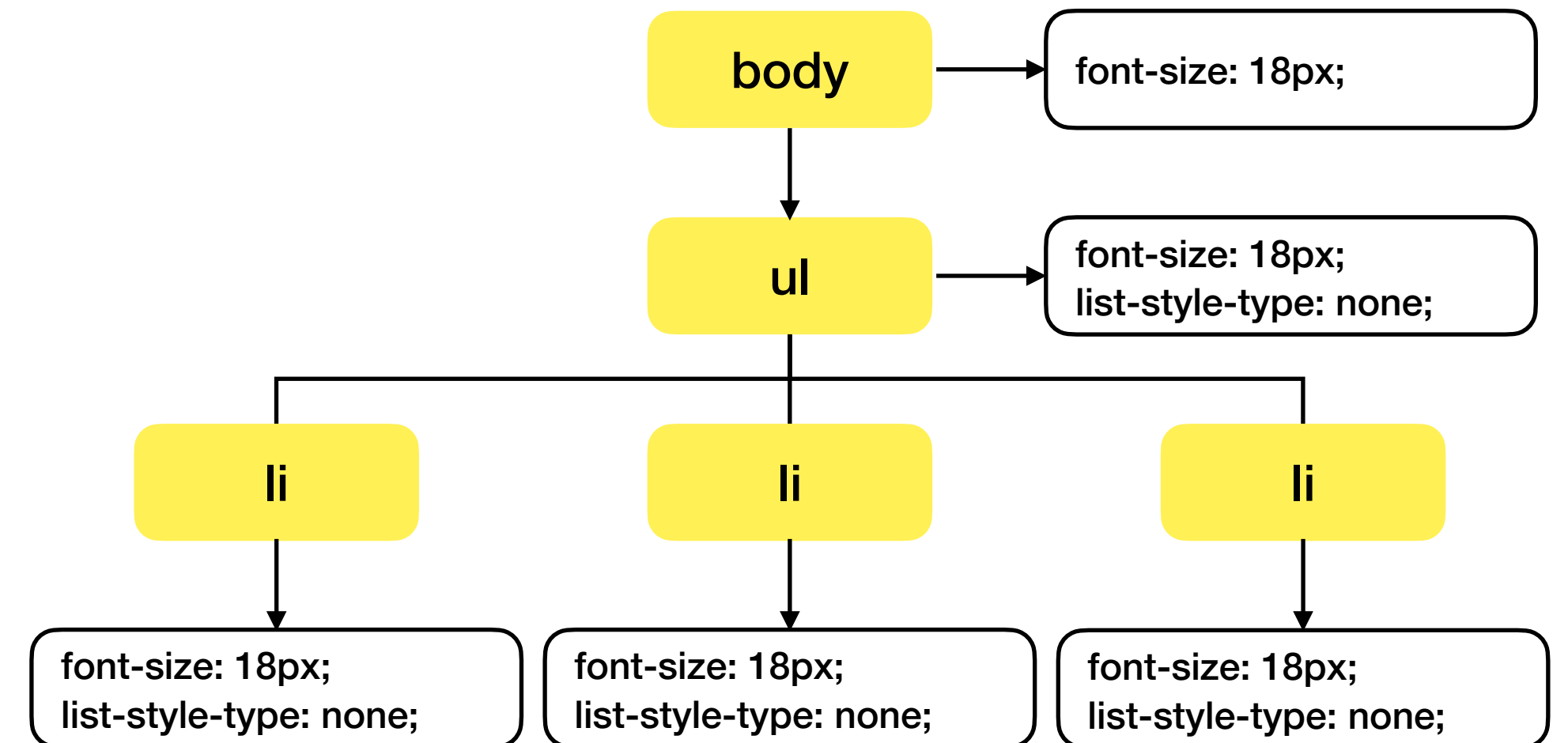
5. 렌더 트리 생성

DOM



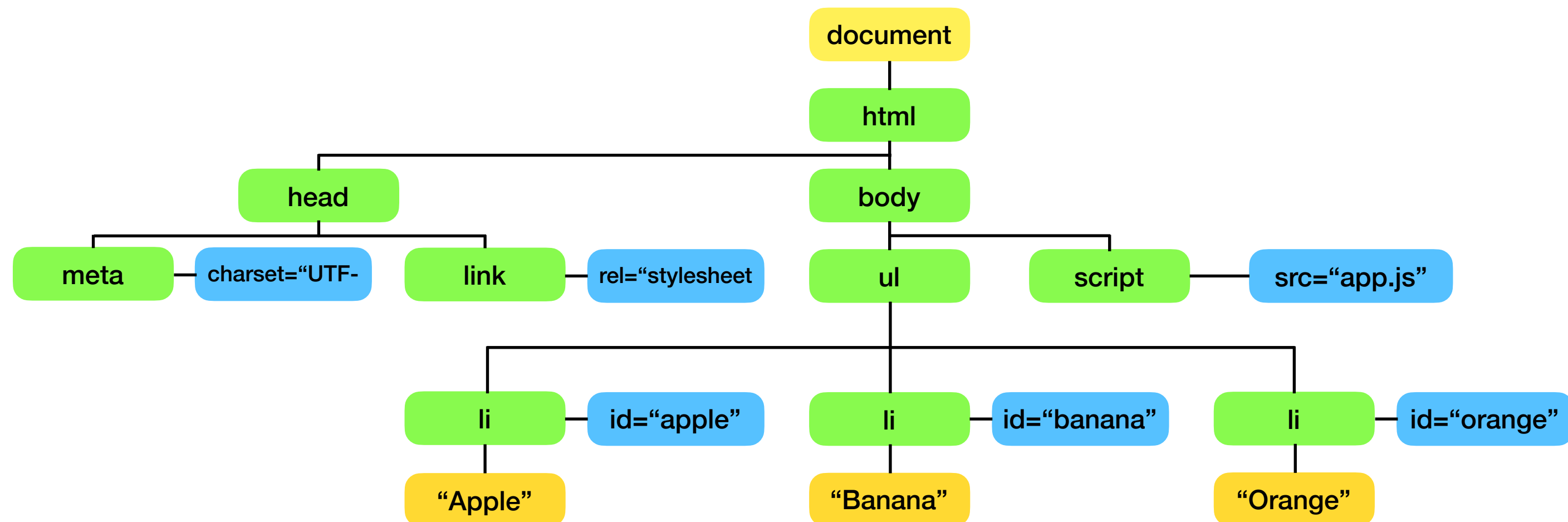
+

CSSOM



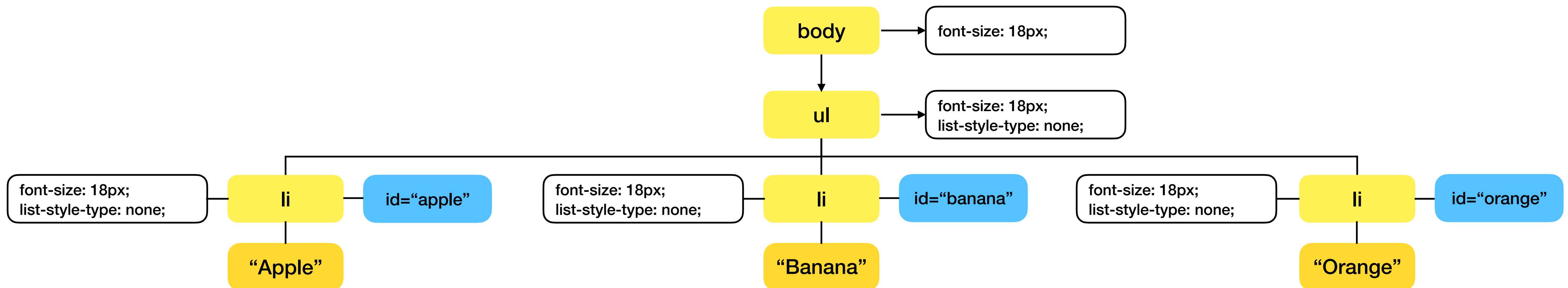
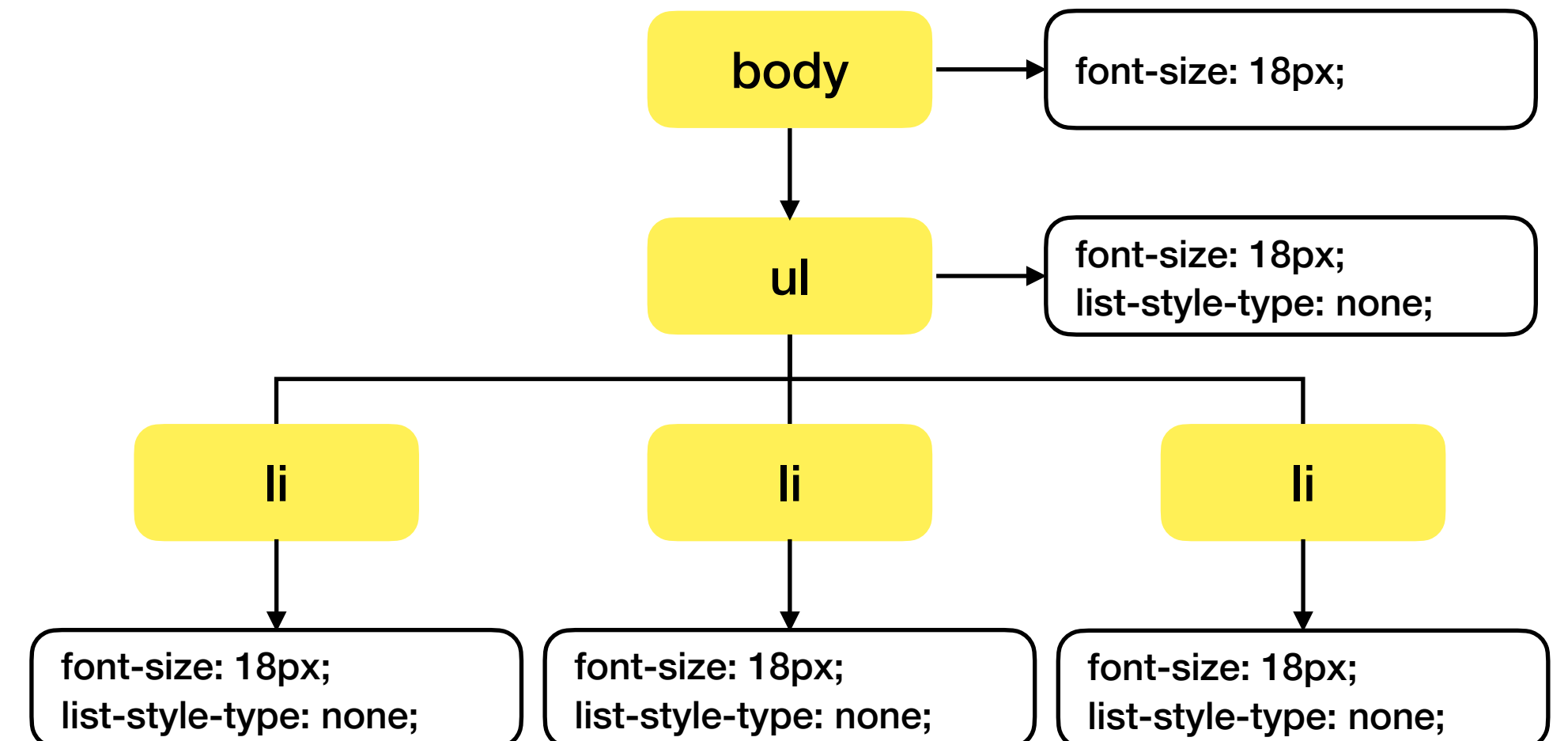
5. 렌더 트리 생성

DOM

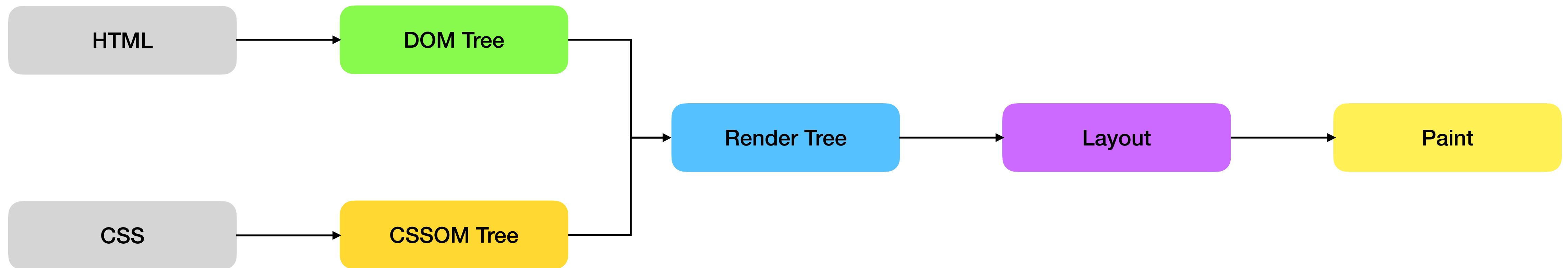


+

CSSOM

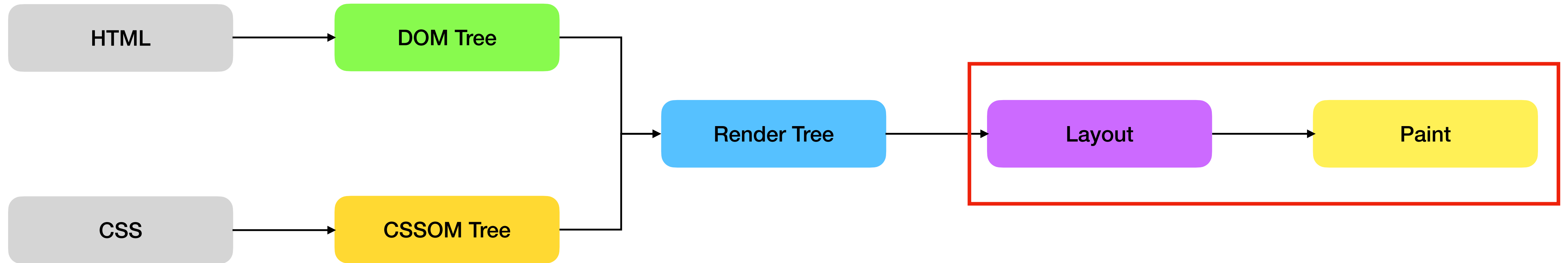


5. 렌더 트리 생성



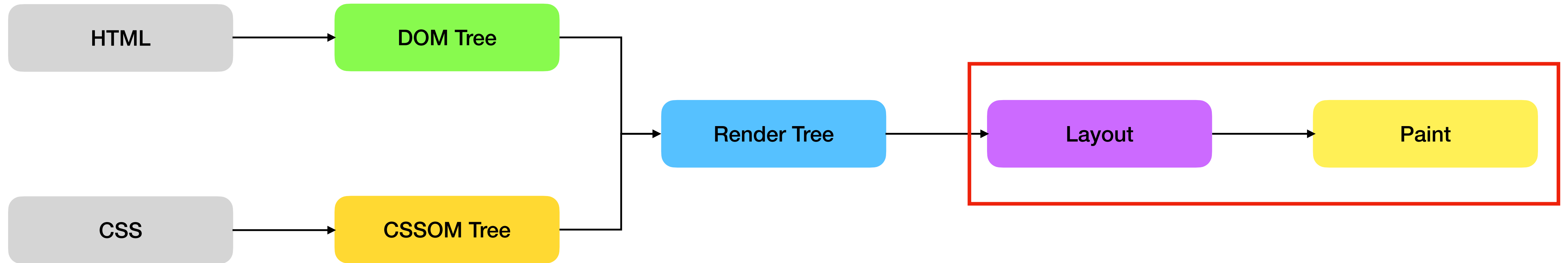
완성된 렌더 트리는 각 HTML 요소의 레이아웃을 계산하는 데 사용되며,
브라우저 화면에 픽셀을 렌더링하는 페인팅 처리에 입력된다.

5. 렌더 트리 생성



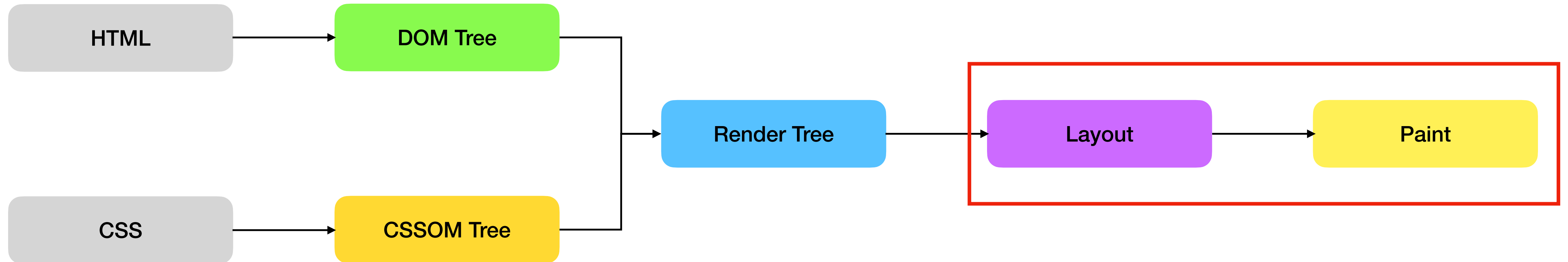
1. 자바스크립트에 의한 노드 추가 또는 삭제

5. 렌더 트리 생성



1. 자바스크립트에 의한 노드 추가 또는 삭제
2. 브라우저 창의 리사이징에 의한 뷰포트 크기 변경

5. 렌더 트리 생성

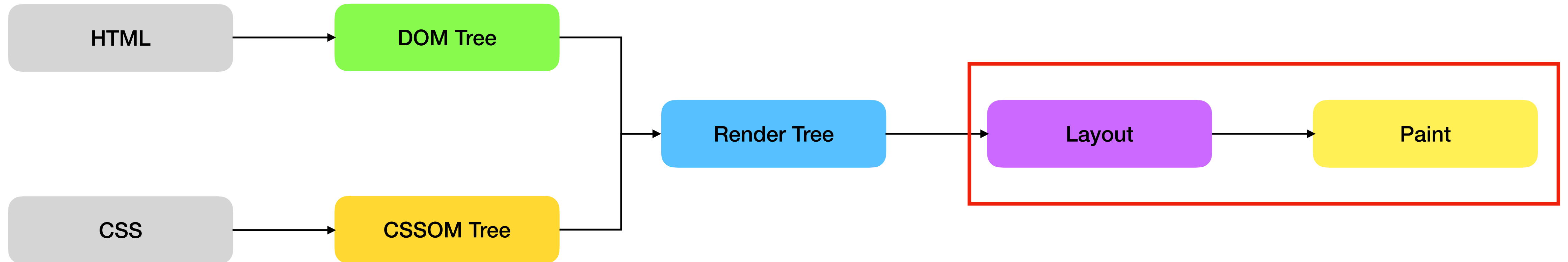


1. 자바스크립트에 의한 노드 추가 또는 삭제

2. 브라우저 창의 리사이징에 의한 뷰포트 크기 변경

3. HTML 요소의 레이아웃에 변경을 발생시키는 스타일의 변경
width/height, margin, padding, border, display, position, top/right/bottom/left 등

5. 렌더 트리 생성



1. 자바스크립트에 의한 노드 추가 또는 삭제

2. 브라우저 창의 리사이징에 의한 뷰포트 크기 변경

3. HTML 요소의 레이아웃에 변경을 발생시키는 스타일의 변경

width/height, margin, padding, border, display, position, top/right/bottom/left 등

레이아웃 계산 및 페인팅을 다시 실행하는 리렌더링은 비용이 많이 드는, 즉 성능에 악영향을 주는 작업이다.
따라서 가급적 리렌더링이 빈번하게 발생하지 않도록 주의해야한다.

6. 자바스크립트 파싱과 실행

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

자바스크립트 파싱과 실행은 자바스크립트 엔진이 처리한다.



1. 자바스크립트 코드를
문법적 의미를 갖는 코드의 최소 단위인 토큰으로 분해한다.

6. 자바스크립트 파싱과 실행

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

자바스크립트 파싱과 실행은 자바스크립트 엔진이 처리한다.

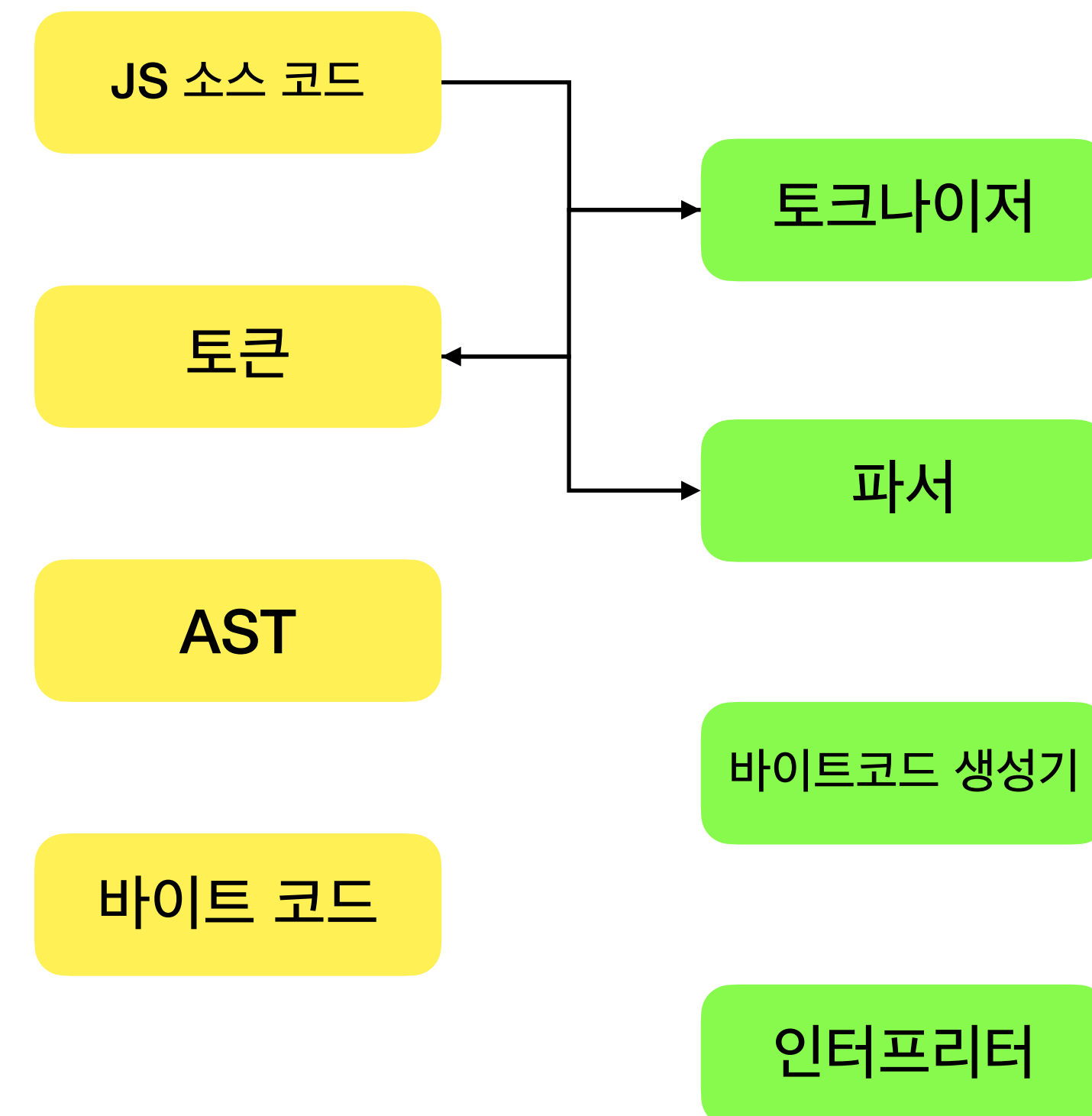


1. 자바스크립트 코드를
문법적 의미를 갖는 코드의 최소 단위인 토큰으로 분해한다.

6. 자바스크립트 파싱과 실행

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

자바스크립트 파싱과 실행은 자바스크립트 엔진이 처리한다.

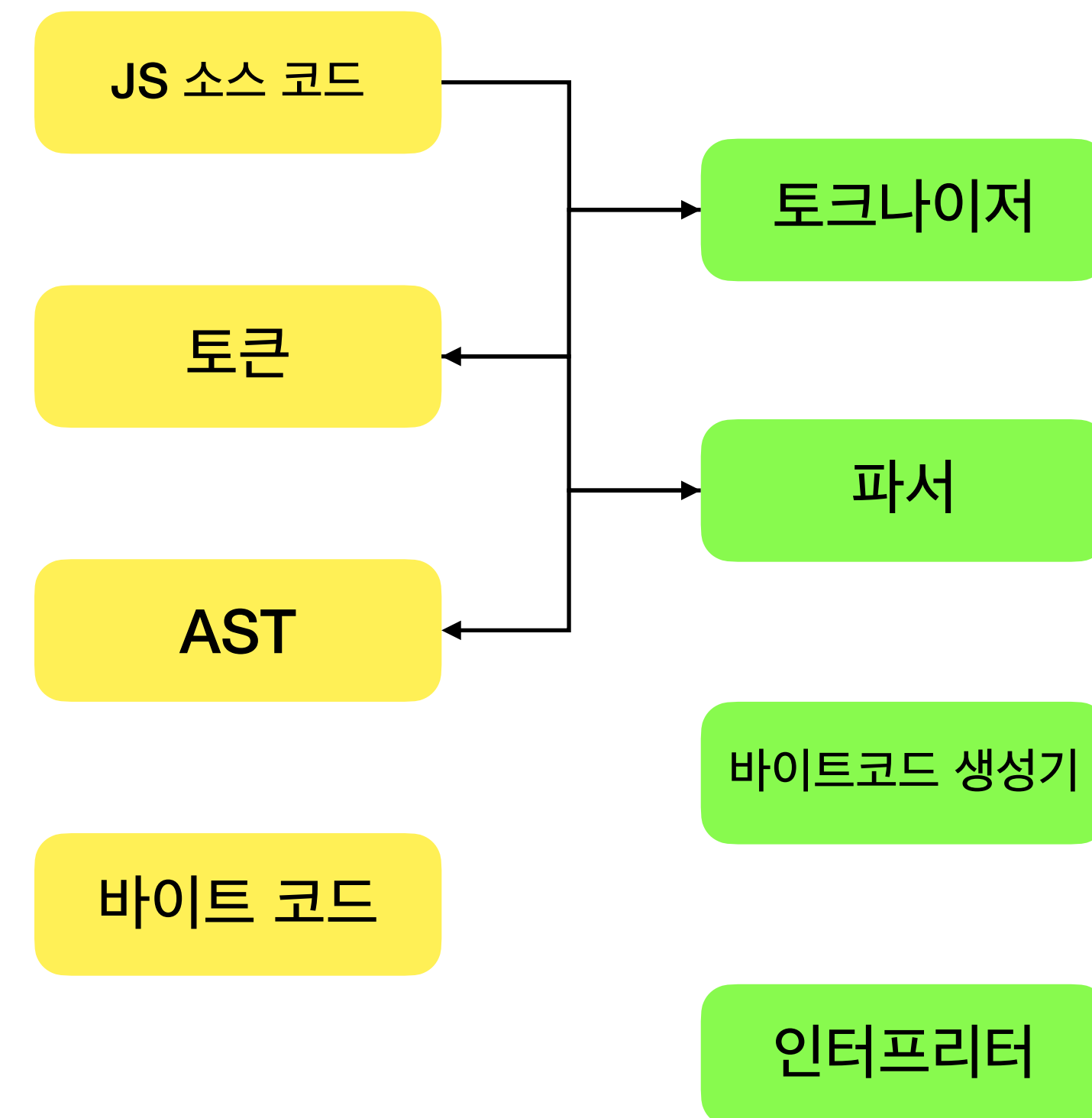


1. 자바스크립트 코드를
문법적 의미를 갖는 코드의 최소 단위인 토큰으로 분해한다.

6. 자바스크립트 파싱과 실행

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

자바스크립트 파싱과 실행은 자바스크립트 엔진이 처리한다.

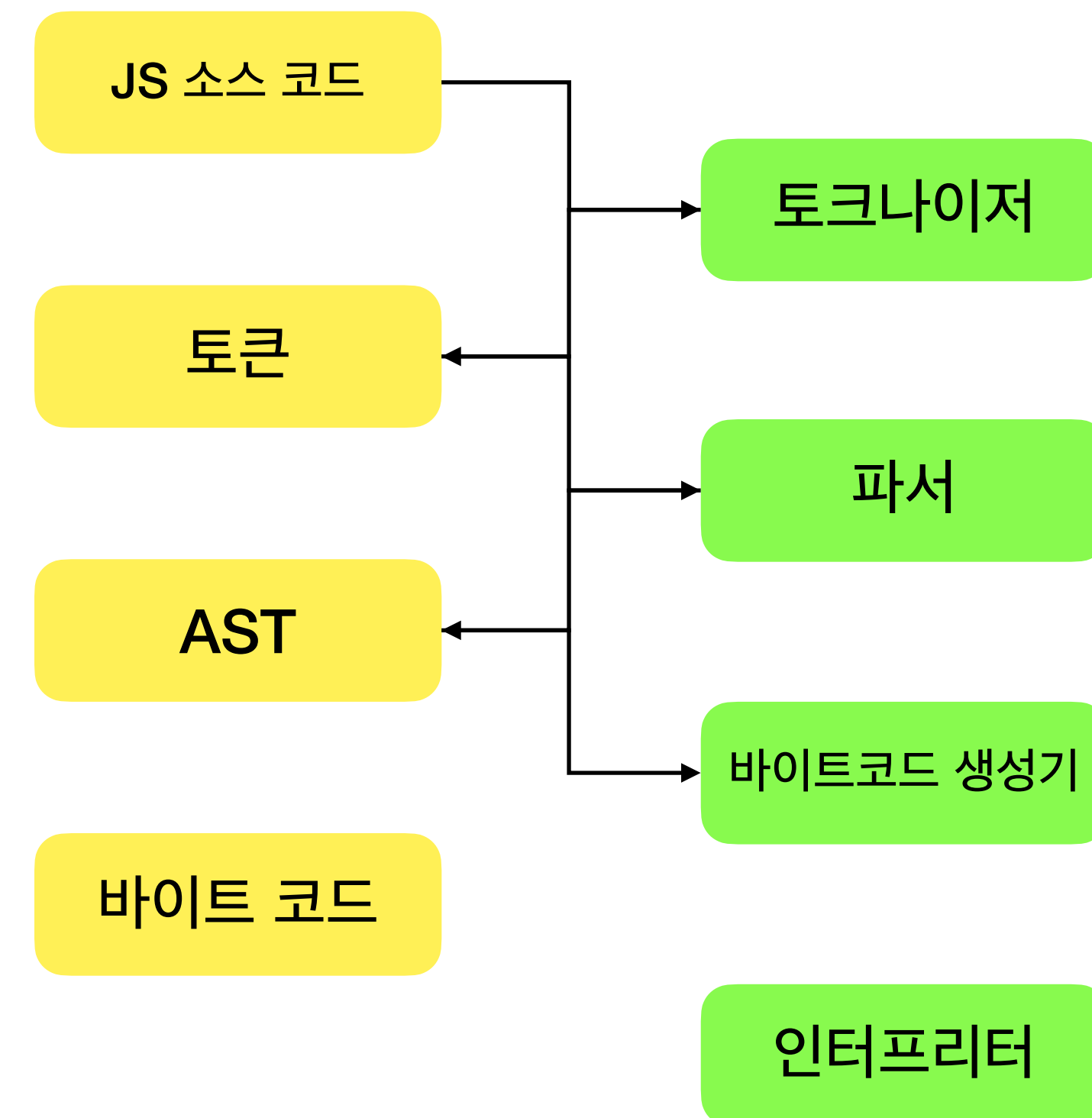


2. 토큰들의 집합을 구문 분석하여 AST를 생성한다.

6. 자바스크립트 파싱과 실행

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

자바스크립트 파싱과 실행은 자바스크립트 엔진이 처리한다.

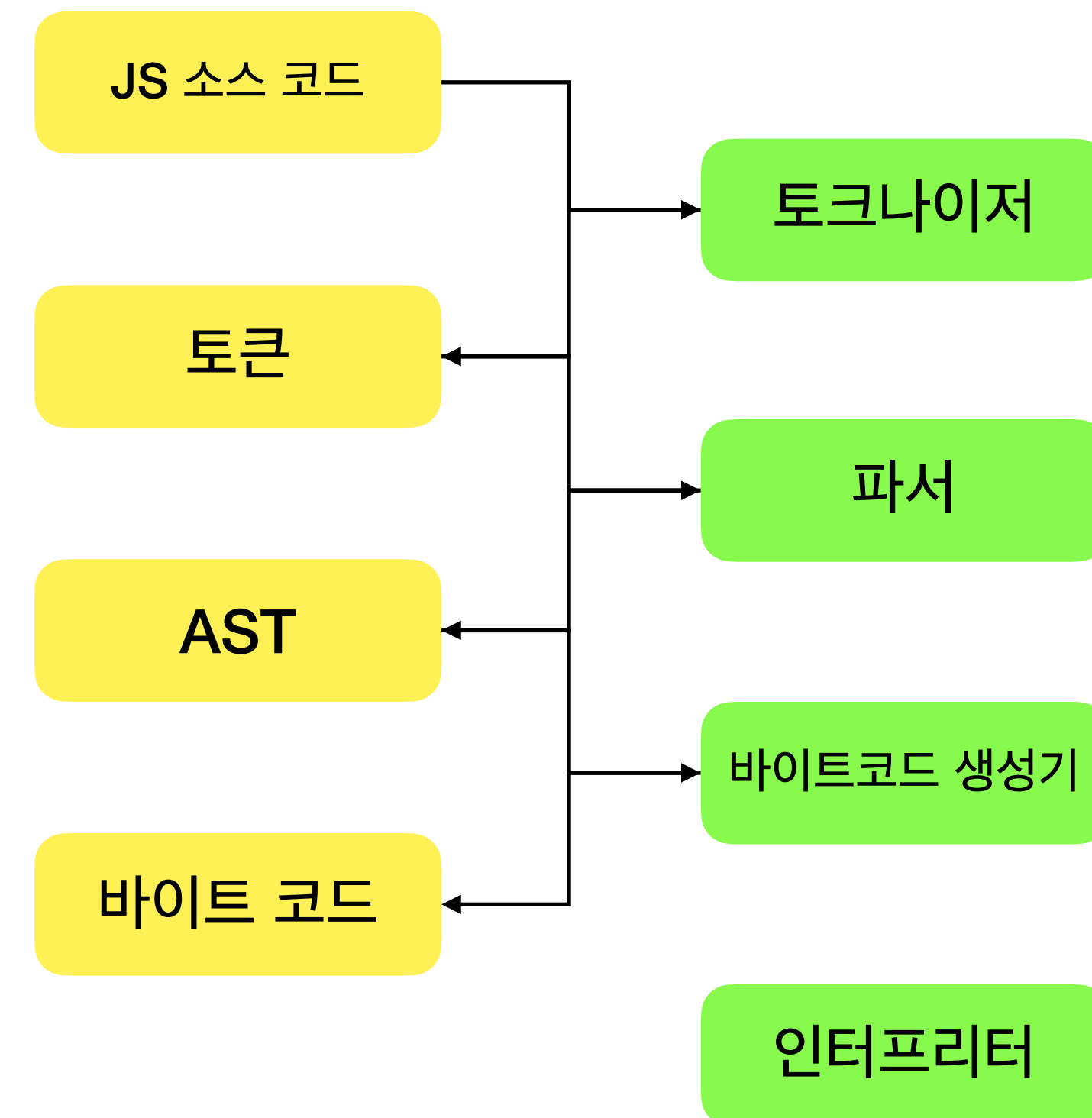


3. AST는 바이트코드로 변환되고
인터프리터에 의해 실행된다.

6. 자바스크립트 파싱과 실행

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

자바스크립트 파싱과 실행은 자바스크립트 엔진이 처리한다.

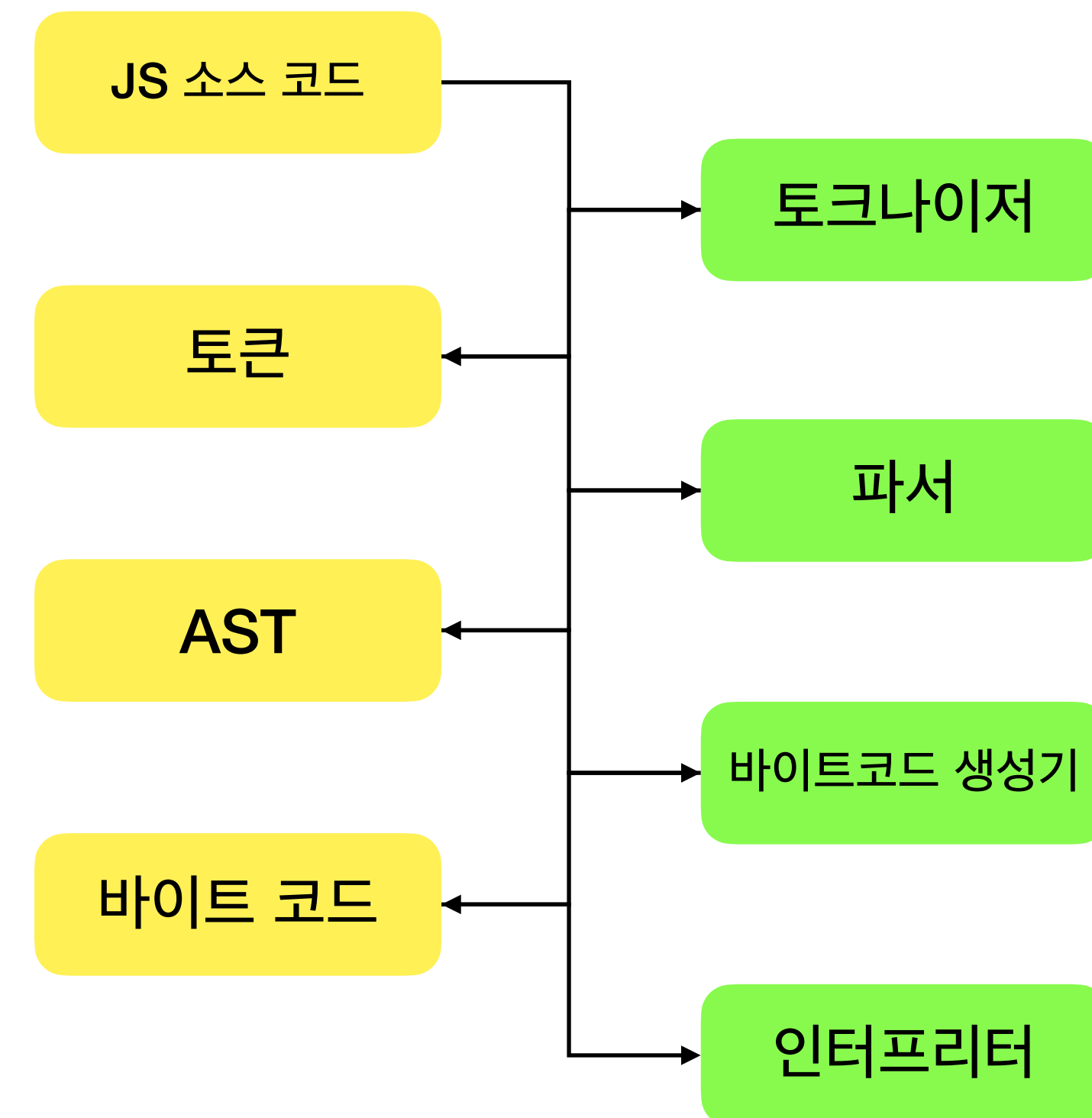


3. AST는 바이트코드로 변환되고
인터프리터에 의해 실행된다.

6. 자바스크립트 파싱과 실행

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

자바스크립트 파싱과 실행은 자바스크립트 엔진이 처리한다.

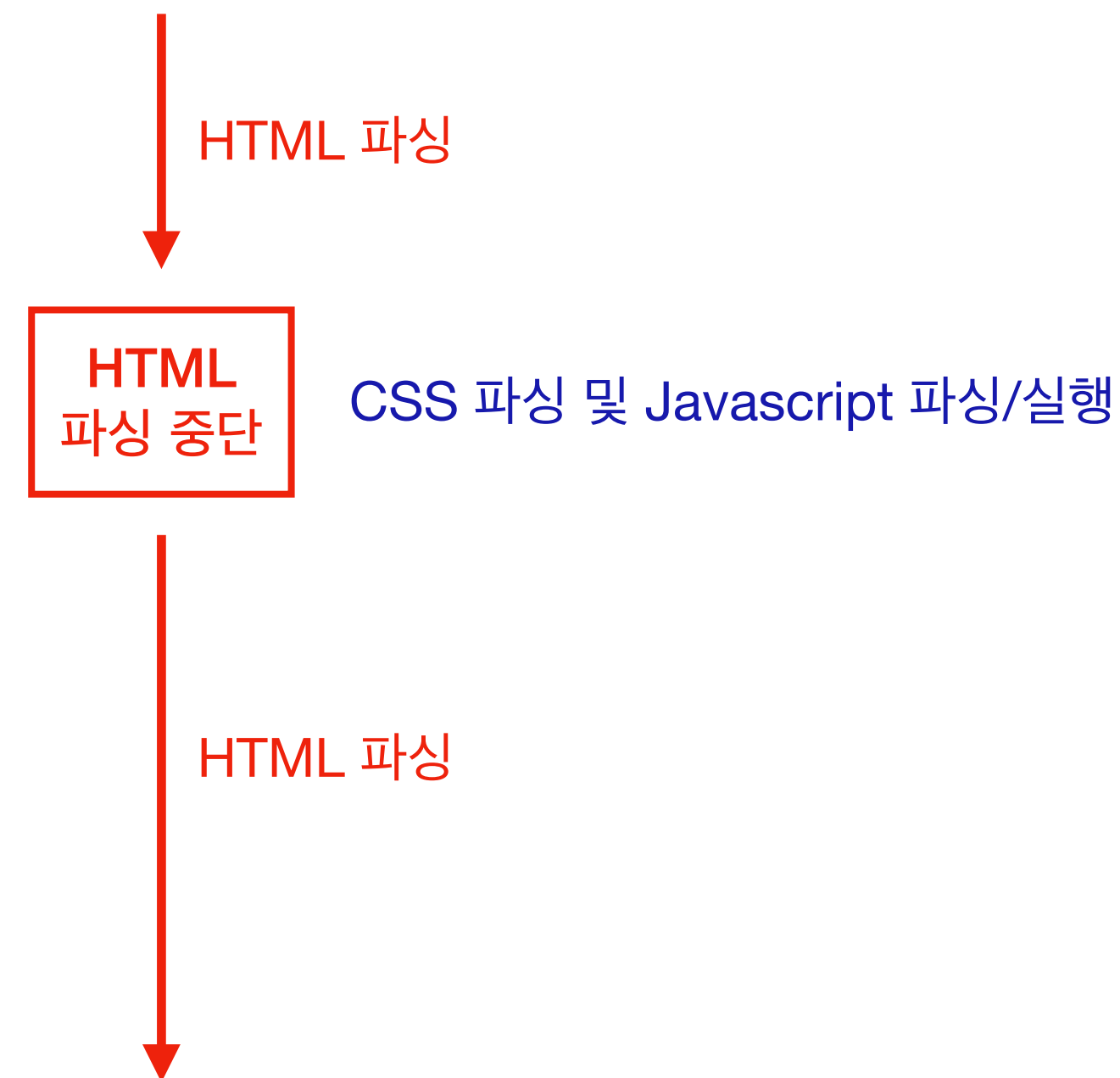


3. AST는 바이트코드로 변환되고
인터프리터에 의해 실행된다.

7. 자바스크립트 파싱에 의한 HTML 파싱 중단

브라우저는 동기적으로, 즉 위에서 아래 방향으로 순차적으로 HTML, CSS, 자바스크립트를 파싱하고 실행한다.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="style.css" />
    <script src="app.js"></script>
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
  </body>
</html>
```



7. 자바스크립트 파싱에 의한 HTML 파싱 중단

자바스크립트를 파싱 및 실행하는 동안
body 태그 안의 내용은 DOM에 존재하지 않는다.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="style.css" />
    <script>
      const $apple = document.getElementById("apple");
      $apple.style.color = "red";
    </script>
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
  </body>
</html>
```

7. 자바스크립트 파싱에 의한 HTML 파싱 중단

자바스크립트를 파싱 및 실행하는 동안
body 태그 안의 내용은 DOM에 존재하지 않는다.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="style.css" />
    <script>
      const $apple = document.getElementById("apple");
      $apple.style.color = "red";
    </script>
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
  </body>
</html>
```

\$apple의 값은 null이 되고 문제가 발생한다.

7. 자바스크립트 파싱에 의한 HTML 파싱 중단



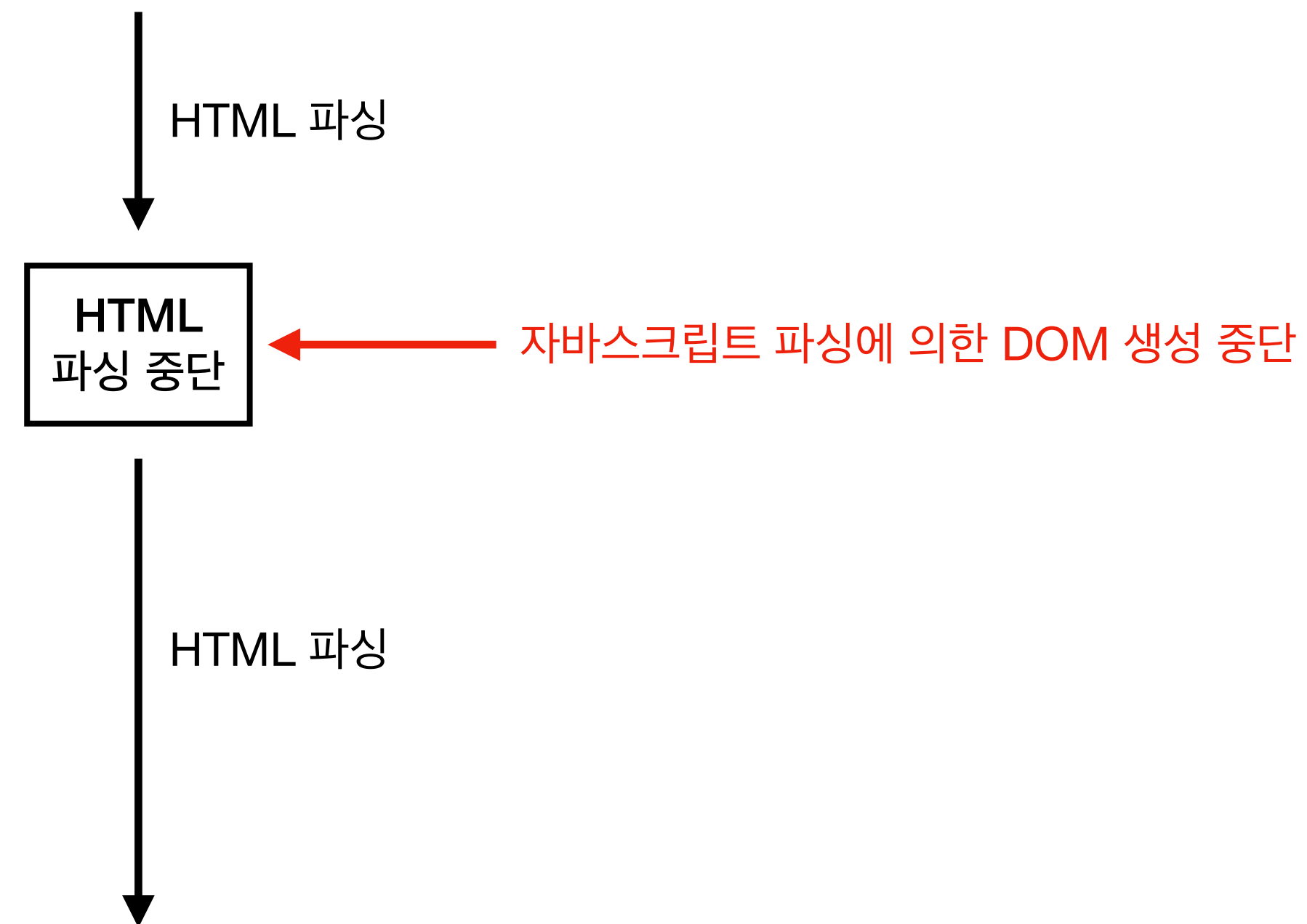
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script>
      const $apple = document.getElementById("apple");
      $apple.style.color = "red";
    </script>
  </body>
</html>
```

자바스크립트를 body 요소의 제일 아래 위치시키면 좋은 점

1. DOM이 미완성인 상태에서 자바스크립트가 DOM을 조작하는 에러가 발생하지 않는다.
2. 자바스크립트가 실행되기 전에 DOM 생성이 완료되어 렌더링되어 로딩시간이 줄어든다.

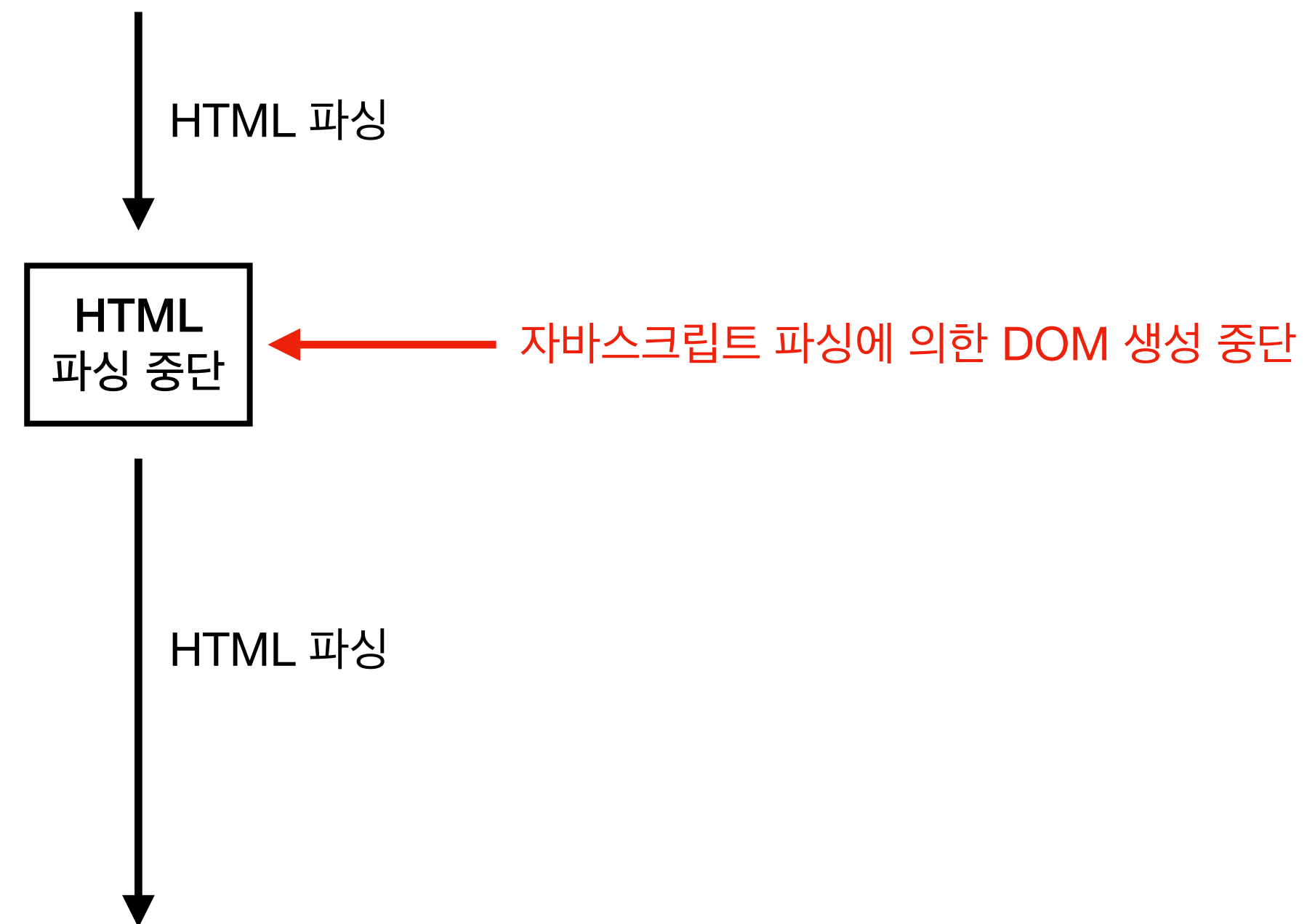
8. script 태그의 async/defer 어트리뷰트

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="style.css" />
    <script src="app.js"></script>
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
  </body>
</html>
```



8. script 태그의 async/defer 어트리뷰트

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="style.css" />
    <script src="app.js"></script>
  </head>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
  </body>
</html>
```



이러한 문제를 해결하기 위해
HTML5부터 script 태그에 async와 defer 어트리뷰트가 추가되었다.

8. script 태그의 async/defer 어트리뷰트

사용시 주의사항

```
<script>
  const $apple = document.getElementById("apple");
  $apple.style.color = "red";
</script>
```

```
<script async src="extern.js"></script>
<script defer src="extern.js"></script>
```

src 어트리뷰트가 없는 인라인 자바스크립트에서는 사용할 수 없다.

8. script 태그의 async/defer 어트리뷰트

사용시 주의사항

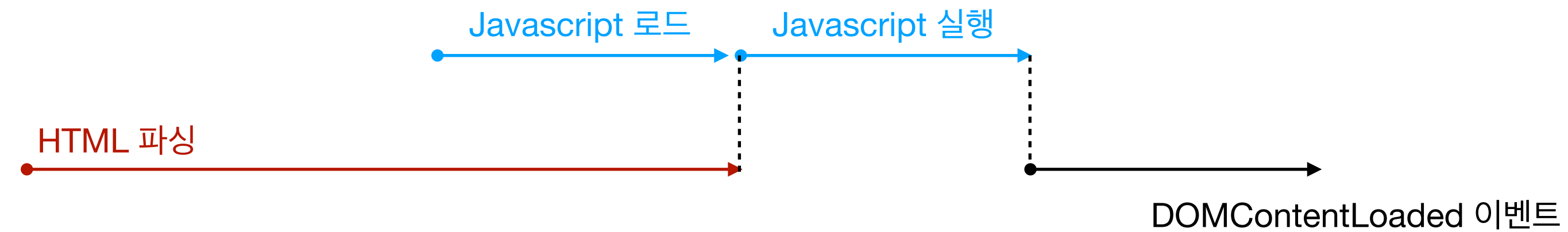
```
<script>  
  const $apple = document.getElementById("apple");  
  $apple.style.color = "red";  
</script>
```

```
<script async src="extern.js"></script>  
<script defer src="extern.js"></script>
```

src 어트리뷰트가 없는 인라인 자바스크립트에서는 사용할 수 없다.

8. script 태그의 async/defer 어트리뷰트

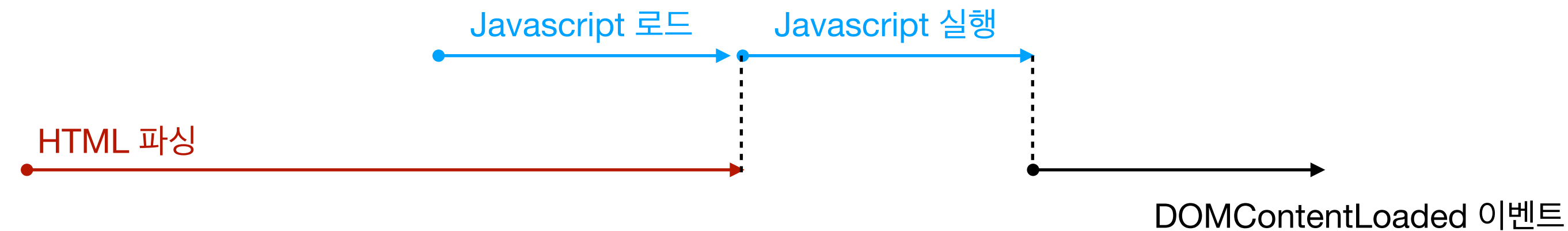
async 어트리뷰트



자바스크립트가 파싱 및 실행되는 동안 HTML 파싱이 중단된다.

8. script 태그의 async/defer 어트리뷰트

async 어트리뷰트

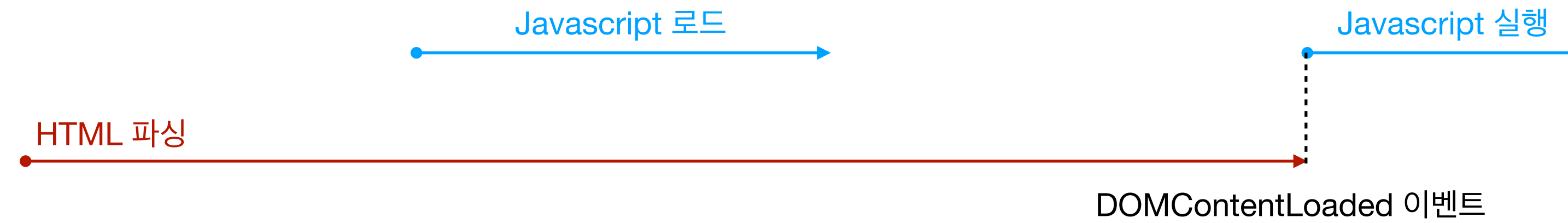


자바스크립트가 파싱 및 실행되는 동안 HTML 파싱이 중단된다.

여러 개의 script 태그에 `async` 어트리뷰트를 지정하면 script 태그 순서와 상관없이 로드가 완료된 자바스크립트부터 실행되므로 순서가 보장되지 않는다.

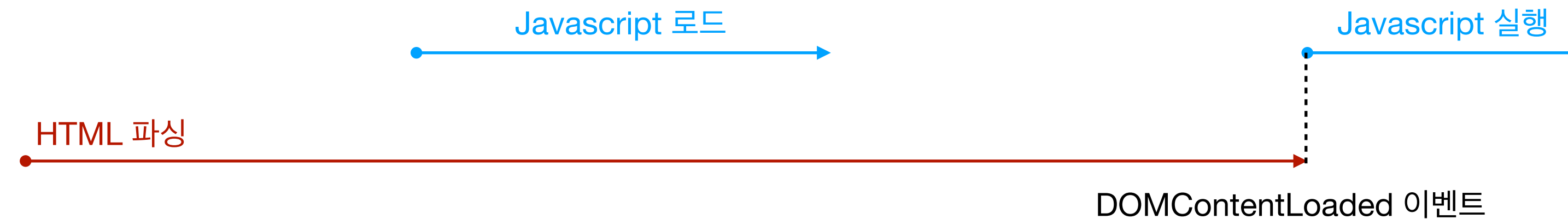
8. script 태그의 async/defer 어트리뷰트

defer 어트리뷰트



8. script 태그의 async/defer 어트리뷰트

defer 어트리뷰트



자바스크립트의 파싱과 실행은 DOM 생성이 완료된 직후에 진행된다.
따라서 DOM 생성이 완료된 이후 실행되어야 할 자바스크립트에 유용하다.