[38장] 브라우저 렌더링 과정

김순요

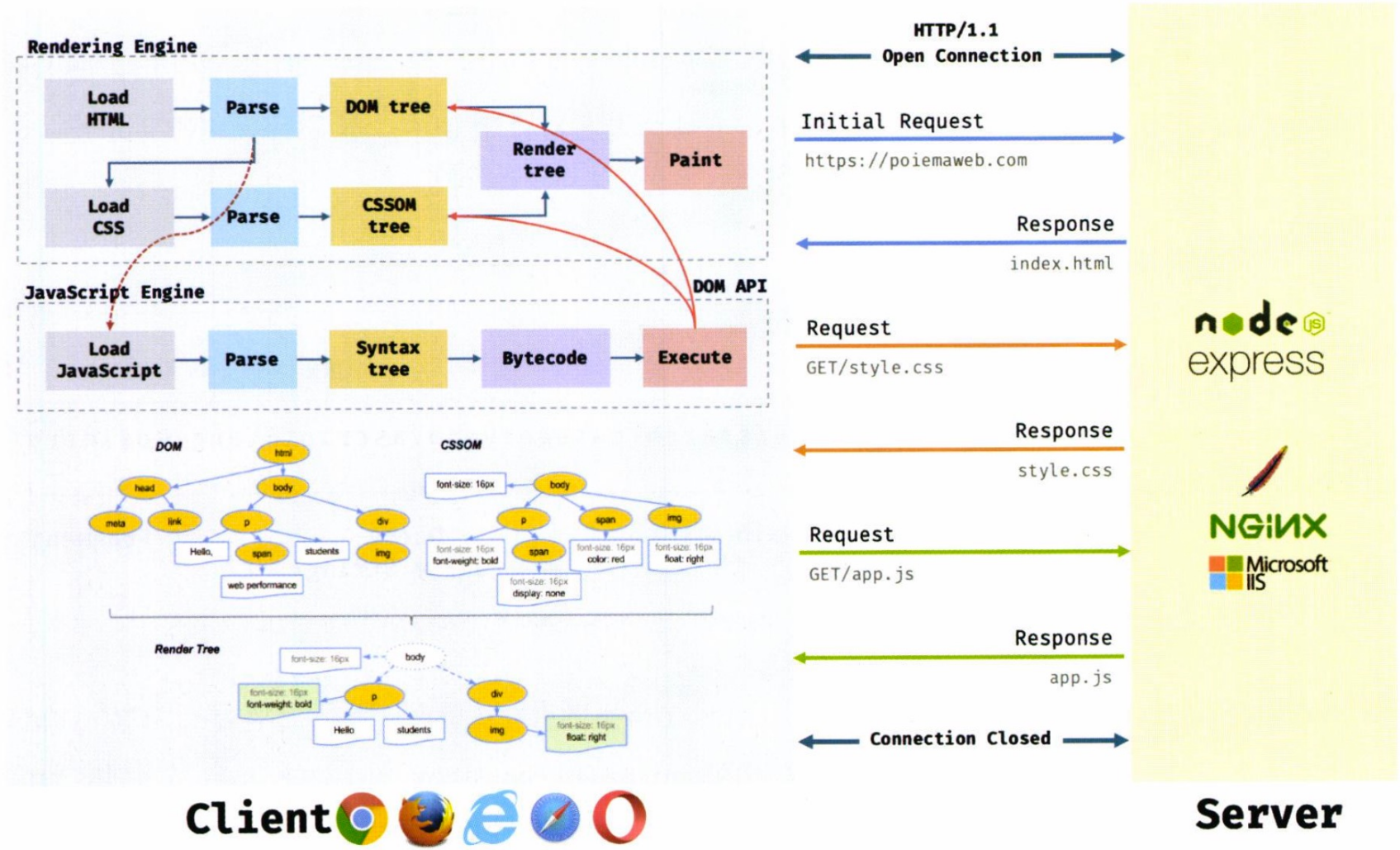


그림 38-1 브라우저의 렌더링 과정

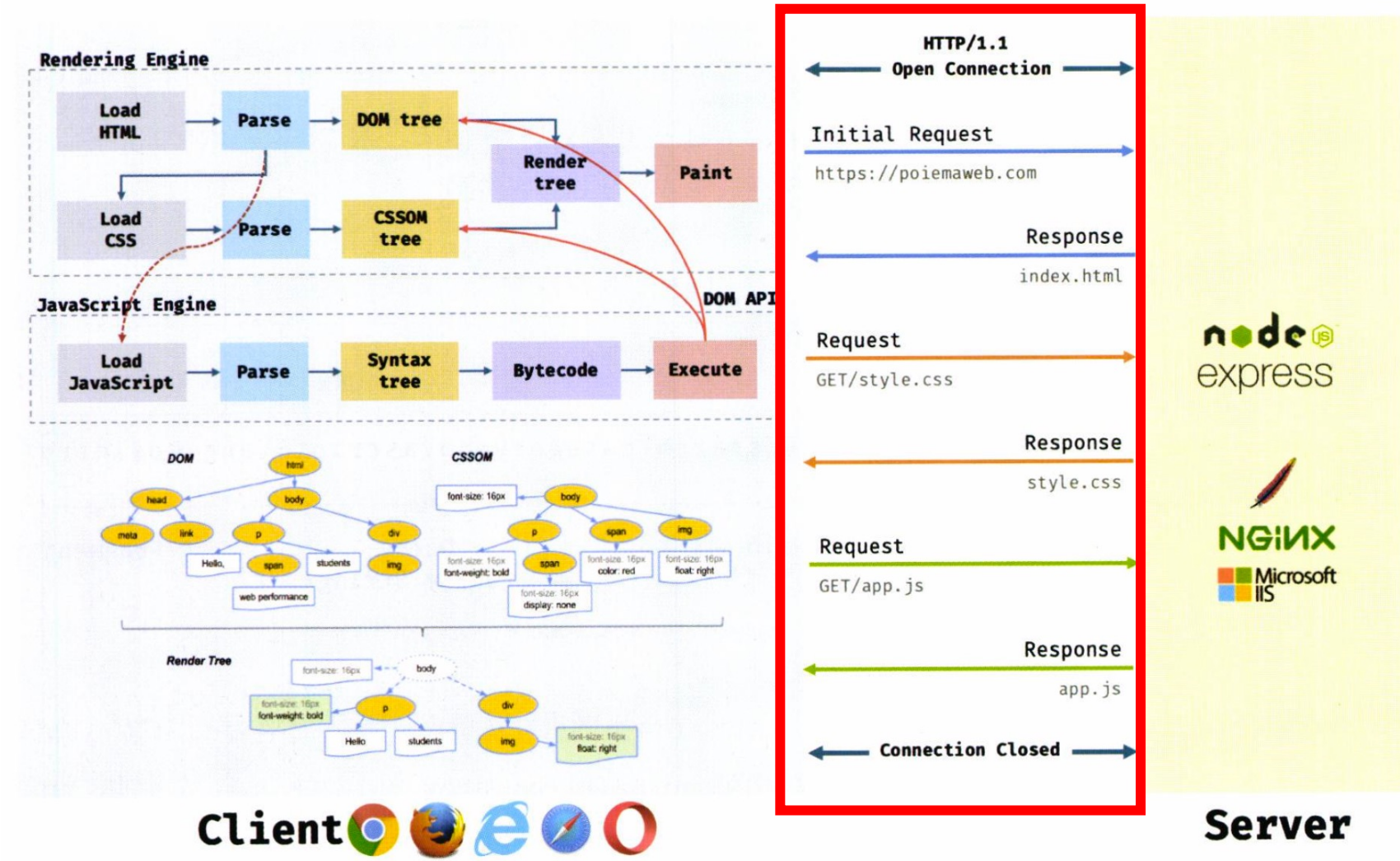


그림 38-1 브라우저의 렌더링 과정

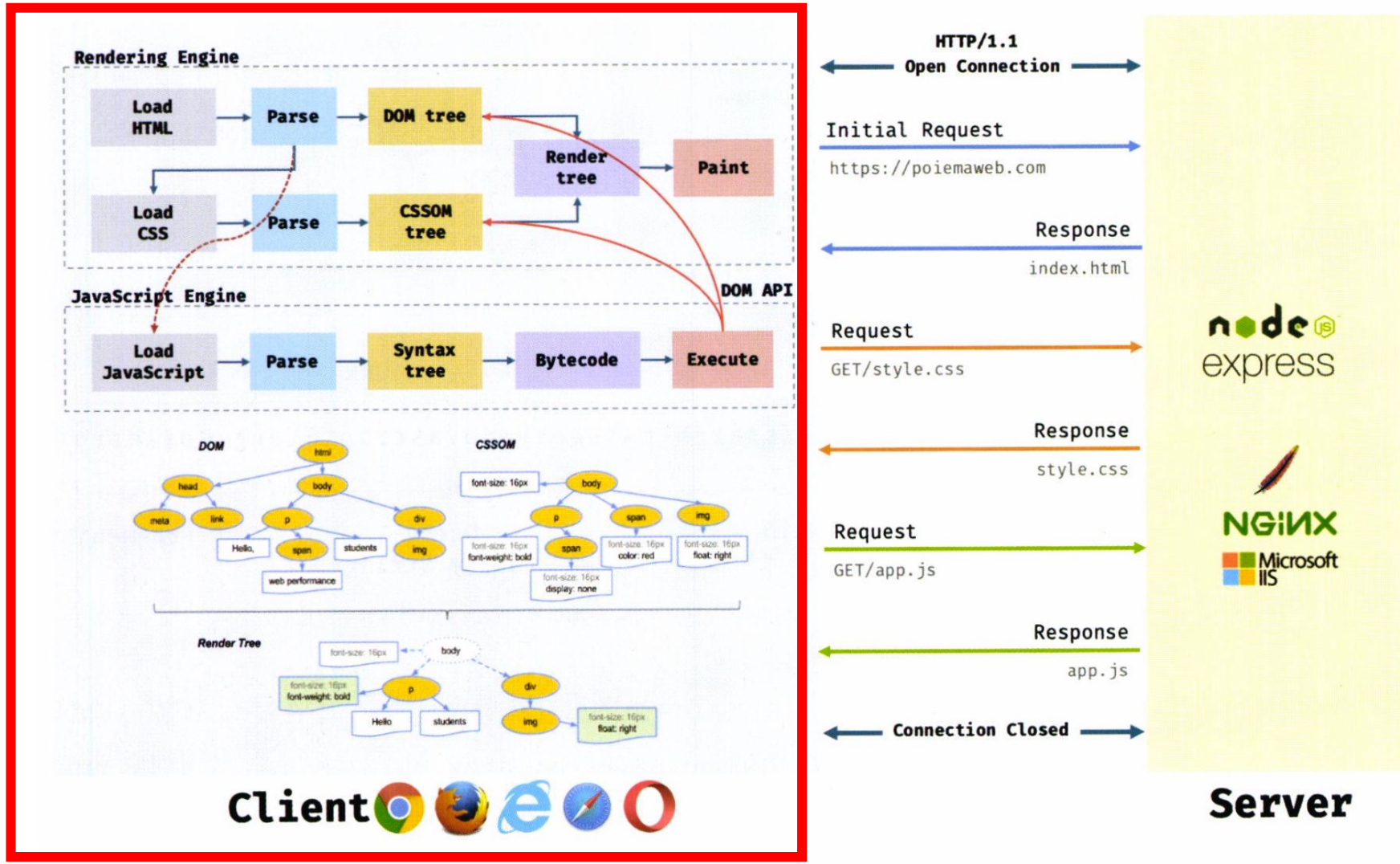


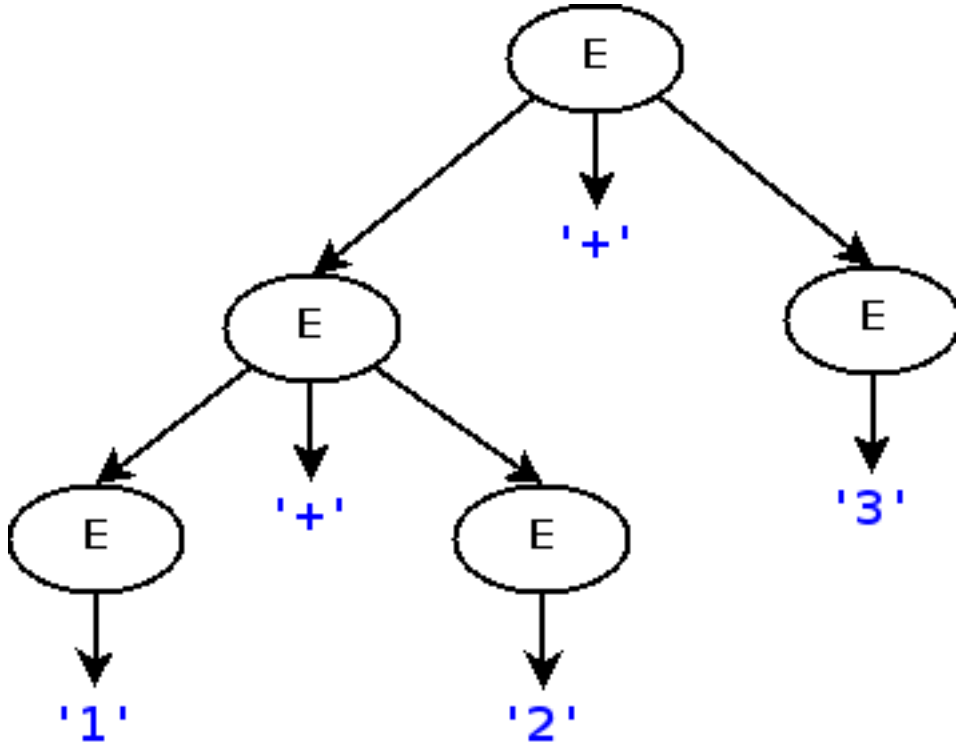
그림 38-1 브라우저의 렌더링 과정

파싱(Parsing)

텍스트 문자열 → 토큰 → 파스 트리

예시) 1 + 2 + 3

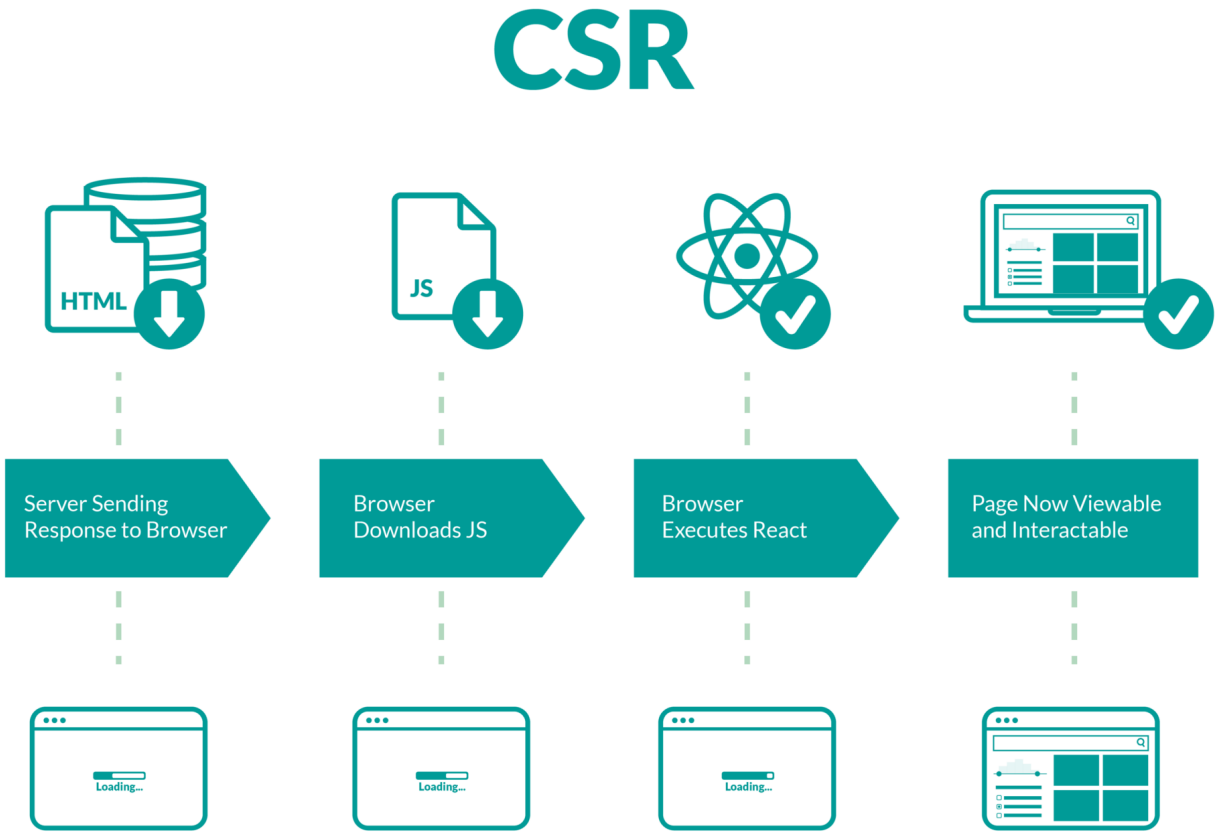
Number: ‘1’, ‘2’, ‘3’
Operation: ‘+’



렌더링(Rendering)

렌더링(Rendering)

HTML, CSS, 자바스크립트로 작성된 문서를 파싱하여 브라우저에 시각적으로 출력하는 것



요청과 응답

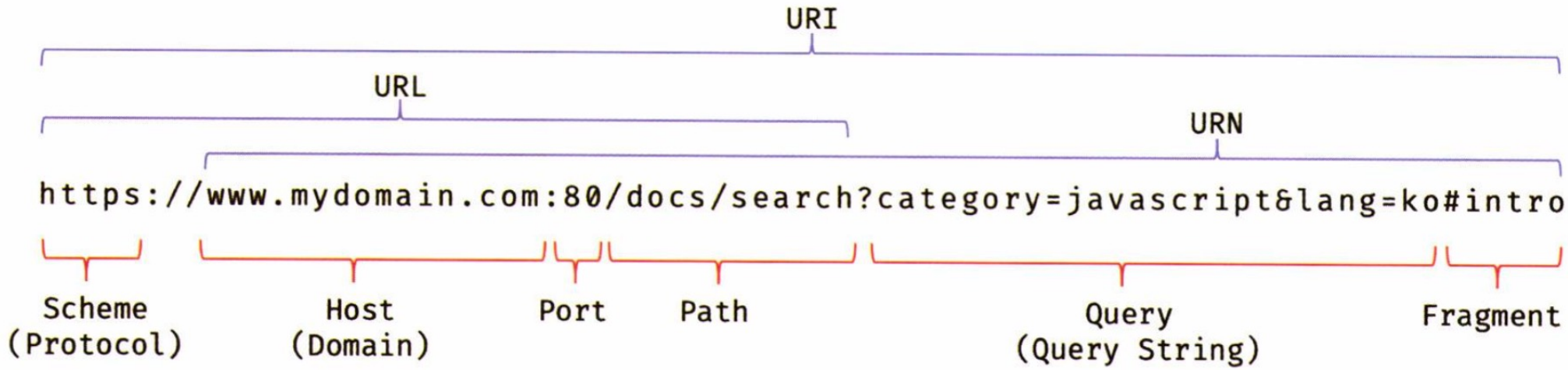
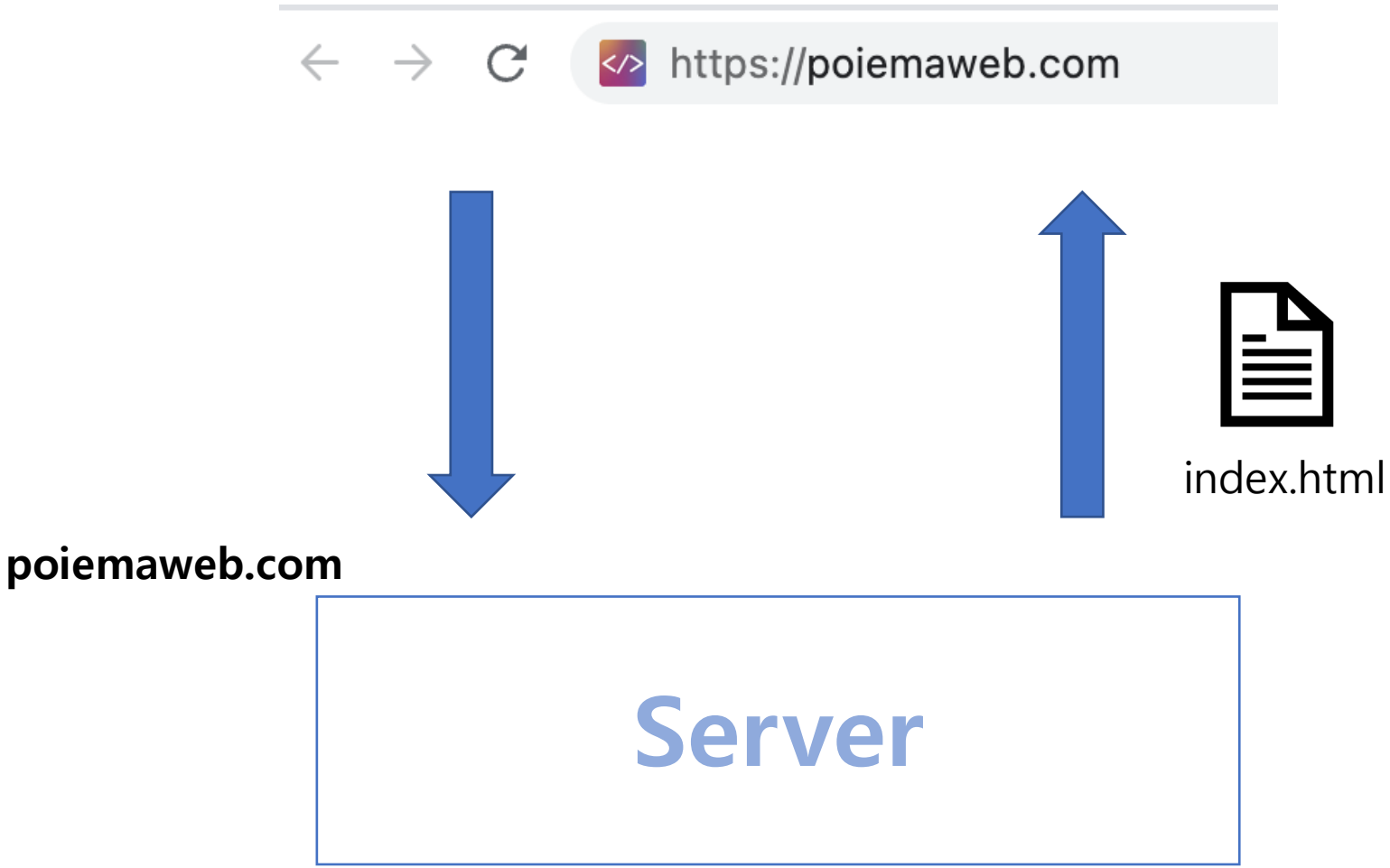
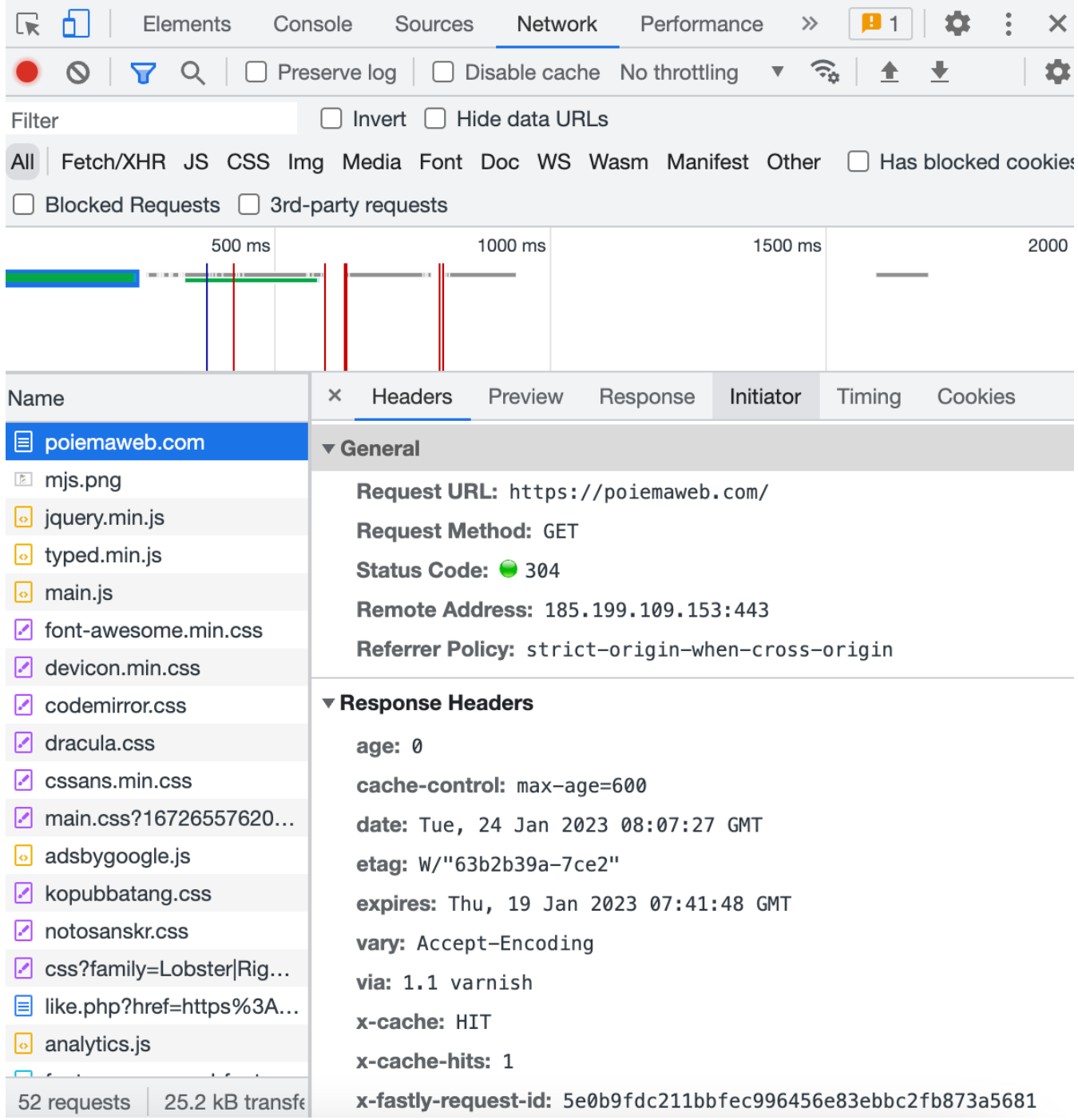
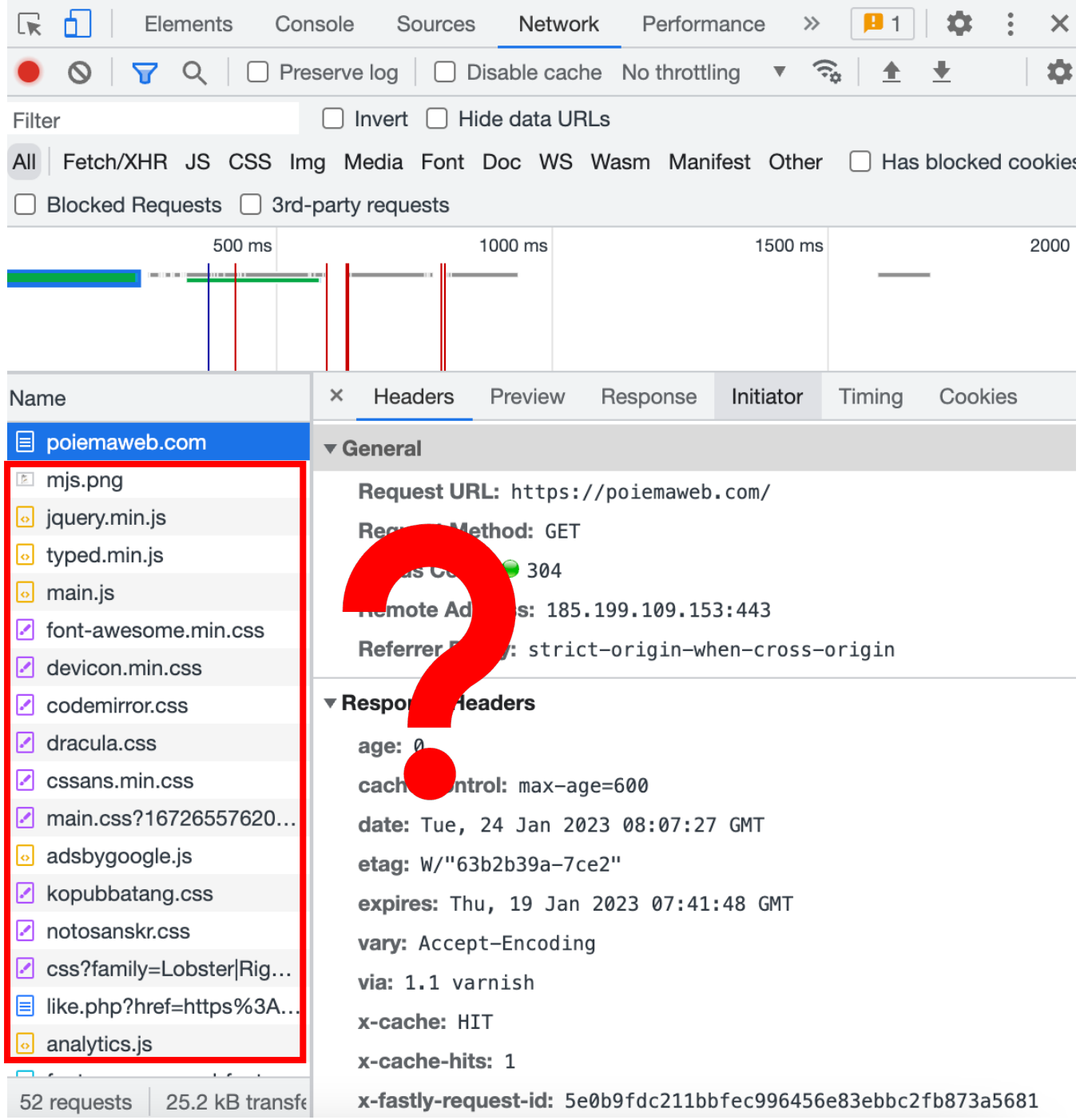


그림 38-2 URI Uniform Resource Identifier 6

요청과 응답




Elements
Console
Sources
Network
Performance
1
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
500 ms
1000 ms
1500 ms
2000
Name
poiemaweb.com
mjs.png
jquery.min.js
typed.min.js
main.js
font-awesome.min.css
devicon.min.css
codemirror.css
dracula.css
cssans.min.css
main.css?16726557620…
adsbygoogle.js
kopubbatang.css
notosanskr.css
css?family=Lobster|Rig…
like.php?href=https%3A…
analytics.js
52 requests
25.2 kB transfe
Headers
Preview
Response
Initiator
Timing
Cookies
General
Request URL: https://poiemaweb.com/
Request Method: GET
Status Code: 304
Remote Address: 185.199.109.153:443
Referrer Policy: strict-origin-when-cross-origin
Response Headers
age: 0
cache-control: max-age=600
date: Tue, 24 Jan 2023 08:07:27 GMT
etag: W/"63b2b39a-7ce2"
expires: Thu, 19 Jan 2023 07:41:48 GMT
vary: Accept-Encoding
via: 1.1 varnish
x-cache: HIT
x-cache-hits: 1
x-fastly-request-id: 5e0b9fdc211bbfec996456e83ebbc2fb873a5681


Elements
Console
Sources
Network
Performance
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
500 ms
1000 ms
1500 ms
2000
Name
poiemaweb.com
mjs.png
jquery.min.js
typed.min.js
main.js
font-awesome.min.css
devicon.min.css
codemirror.css
dracula.css
cssans.min.css
main.css?16726557620...
adsbygoogle.js
kopubbatang.css
notosanskr.css
css?family=Lobster|Rig...
like.php?href=https%3A...
analytics.js
52 requests
25.2 kB transfe
Headers
Preview
Response
Initiator
Timing
Cookies
General
Request URL: https://poiemaweb.com/
304
185.199.109.153:443
strict-origin-when-cross-origin
age: 0
max-age=600
date: Tue, 24 Jan 2023 08:07:27 GMT
etag: W/"63b2b39a-7ce2"
expires: Thu, 19 Jan 2023 07:41:48 GMT
vary: Accept-Encoding
via: 1.1 varnish
x-cache: HIT
x-cache-hits: 1
x-fastly-request-id: 5e0b9fdc211bbfec996456e83ebbc2fb873a5681

```
<link rel="icon" href="https://poiemaweb.com/img/poiemaweb.jpg">
<link rel="stylesheet" href="/assets/vendor/font-awesome/css/font-awesome.min.css">
<link rel="stylesheet" href="/assets/vendor/devicon/devicon.min.css">
<link rel="stylesheet" href="/assets/vendor/codemirror/codemirror.css">
<link rel="stylesheet" href="/assets/vendor/codemirror/theme/dracula.css">
<link rel="stylesheet" href="/assets/vendor/cssans/cssans.min.css">
<link rel="stylesheet" href="/css/main.css?16726557620785202 89">
```

렌더링 엔진이 HTML을 파싱하면서

CSS 파일을 로드하는 link 태그

이미지 파일을 로드하는 img 태그

자바스크립트를 로드하는 script 태그 등을 만나면

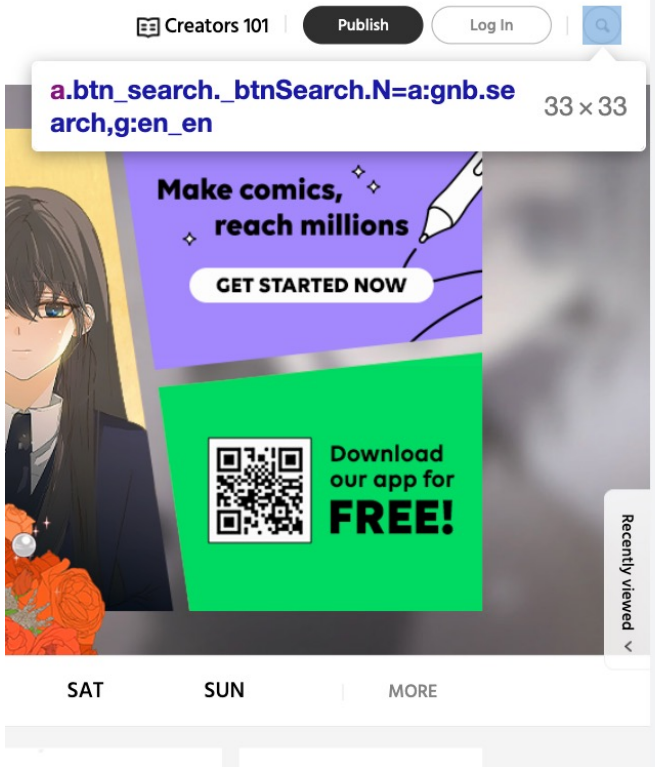
HTML 파싱을 일시 중단하고 해당 리소스 파일을 서버에 요청!

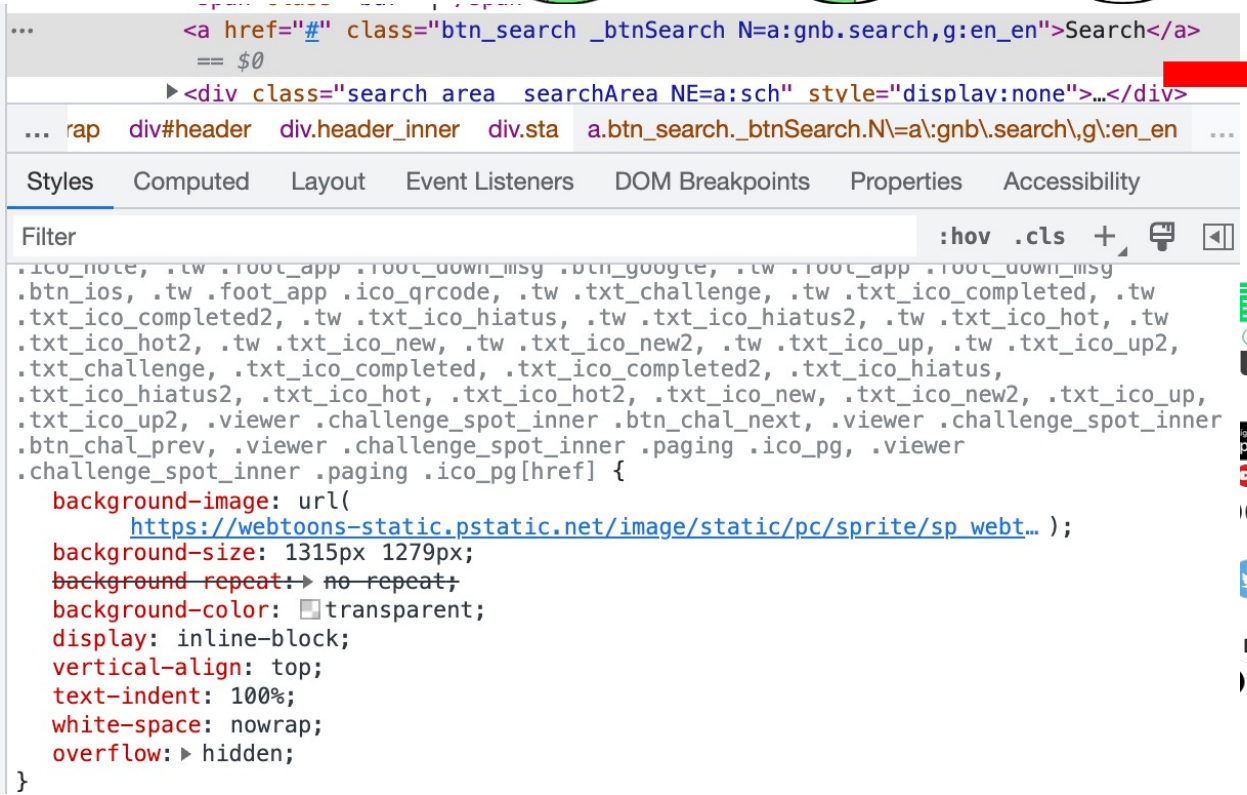
HTTP 1.1과 HTTP2.0

해외 Naver webtoon의 사례)

이미지 스프라이트

아이콘, 이미지를 모아 놓은 정적 파일을 하나를 불러와 위치를 조정하며 이미지 출력





HTTP 1.1

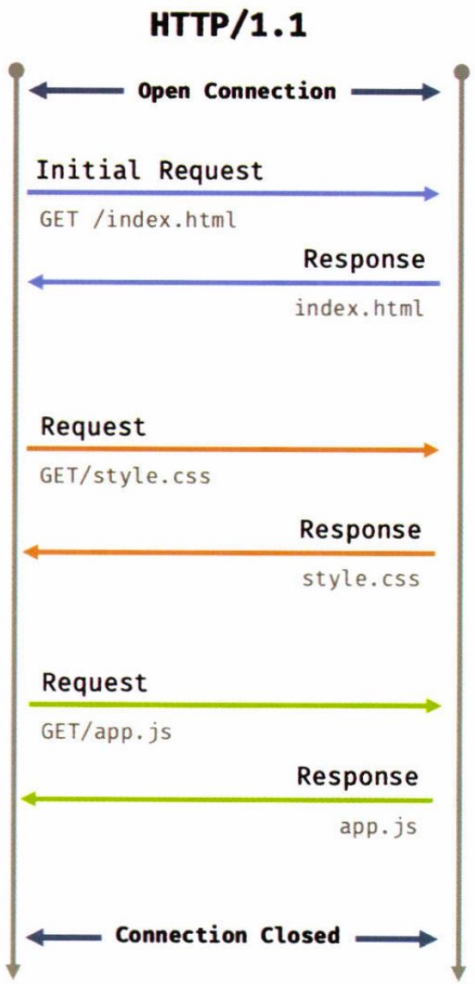


그림 38-4 HTTP/1.1

- 기본적으로 커넥션(connection) 당 하나의 요청과 응답만 처리
- 여러 개의 요청을 한번에 전송할 수 없고, 응답도 마찬가지
- HTML 문서 내에 포함된 여러 개의 리소스 요청(link, img, script 등) 태그에 의한 **요청이 개별적으로 전송되고 응답도 개별적으로 전송**된다.
- **리소스의 동시 전송이 불가능한 구조**이기 때문에 요청할 리소스 개수에 비례하여 응답 시간도 증가한다는 단점이 있음

HTTP 2.0

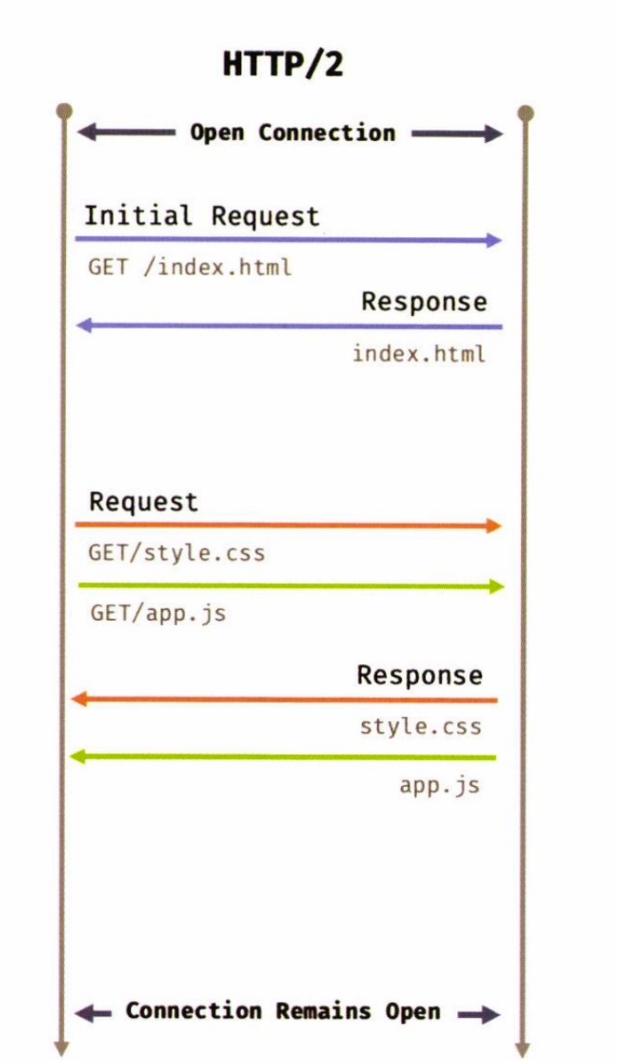


그림 38-5 HTTP/2

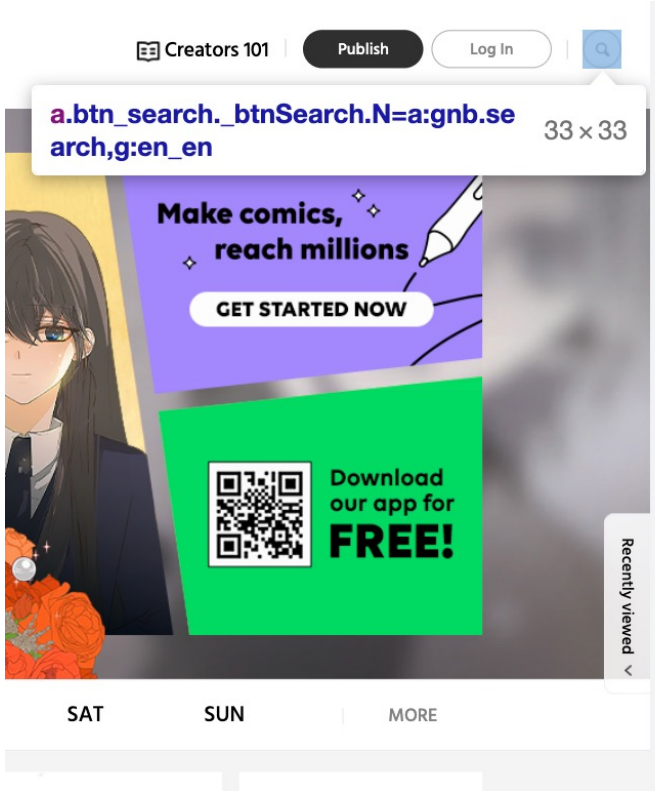
- 커넥션 당 **여러 개의 요청과 응답(다중 요청/응답)**이 가능
- HTTP1.1에 비해 페이지 로드 속도가 약 50% 빠르다고 알려짐

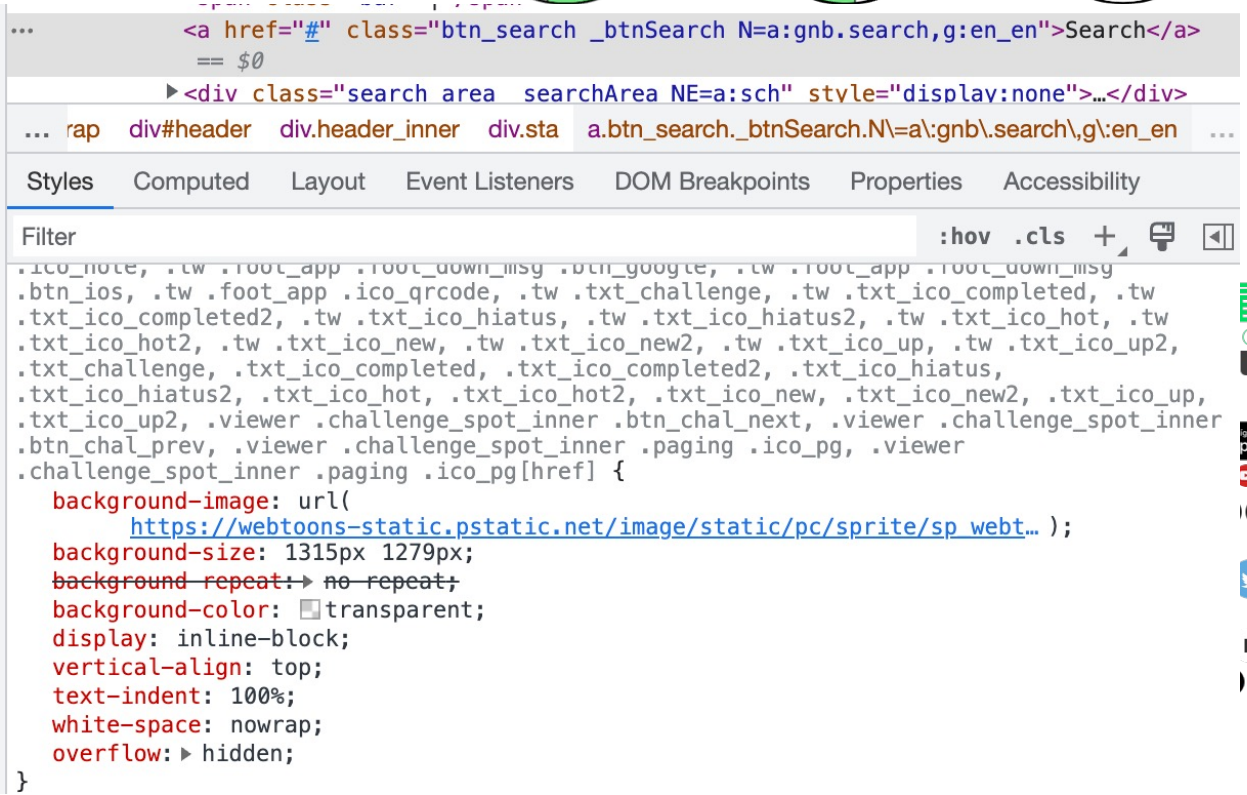
WHY?

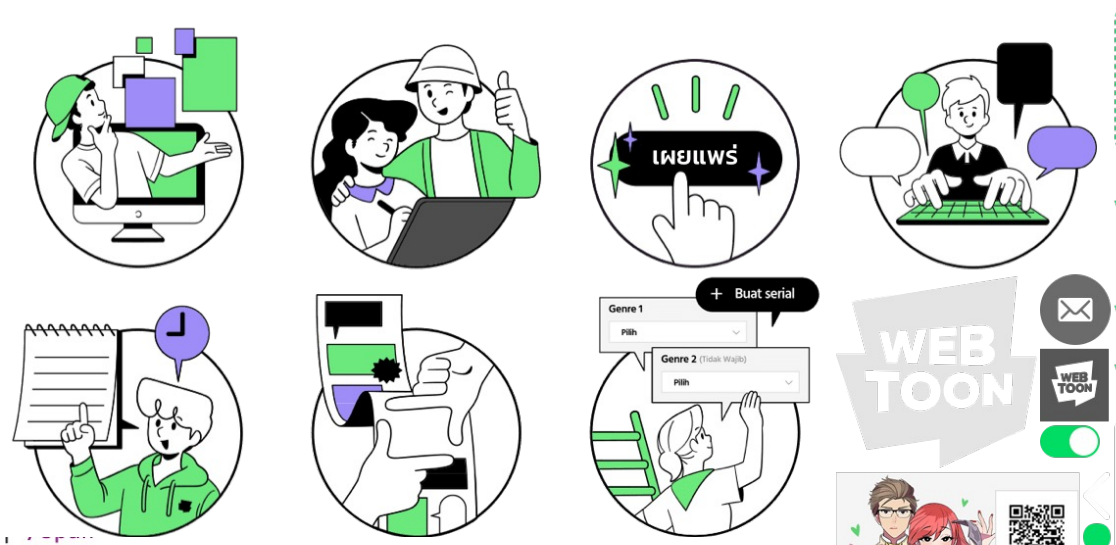
해외 Naver webtoon의 사례)

이미지 스프라이트 방식

아이콘, 이미지를 모아 놓은 정적 파일을 하나를 불러와 위치를 조정







HTML 파싱과 DOM 생성

다음 index.html을 서버로부터 응답받았다고 가정

HTML

```html
<!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>
```

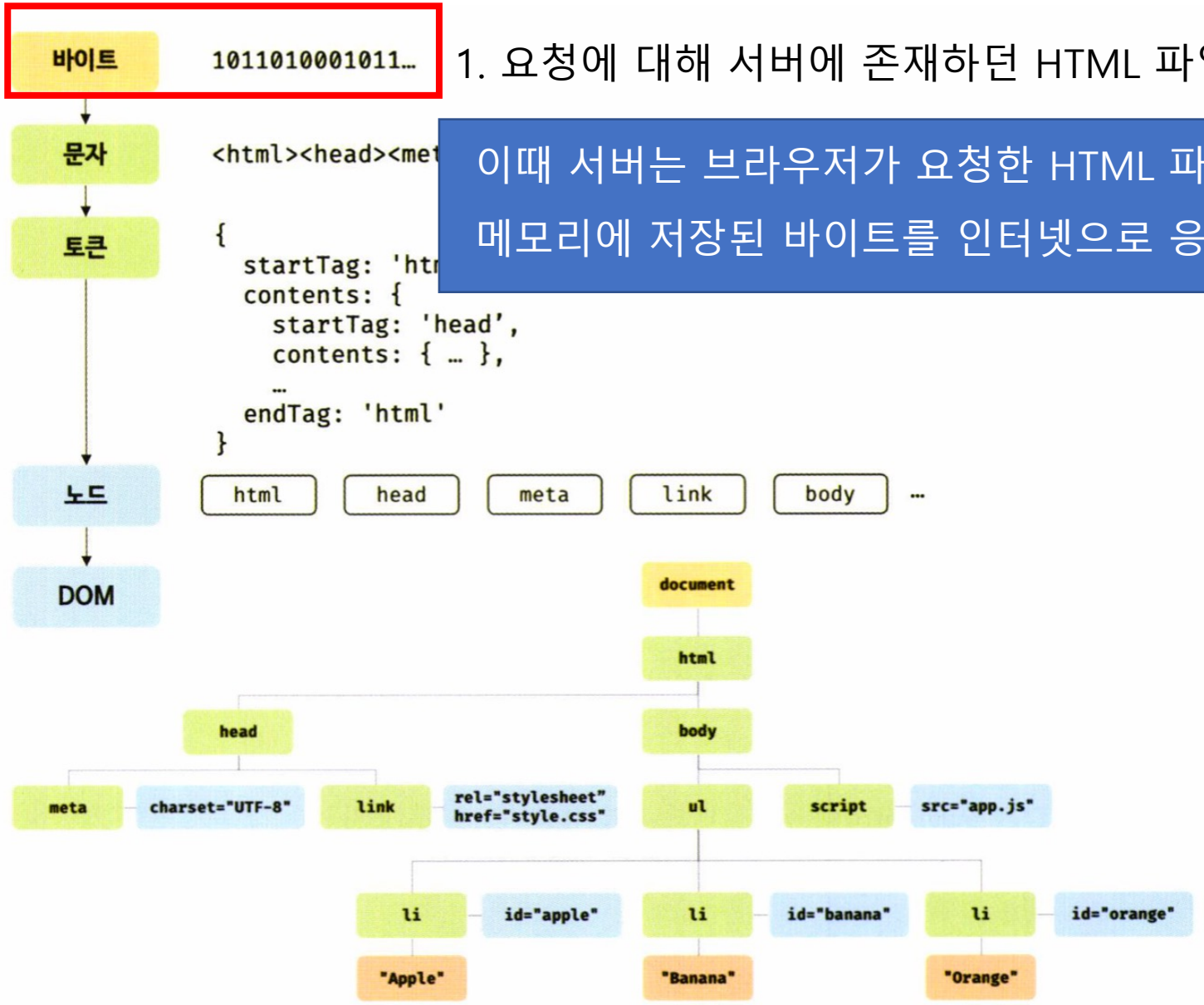


그림 38-6 HTML 파싱과 DOM 생성

1. 요청에 대해 서버에 존재하던 HTML 파일이 응답된다.

이때 서버는 브라우저가 요청한 HTML 파일을 읽어 메모리에 저장한 다음, 메모리에 저장된 바이트를 인터넷으로 응답

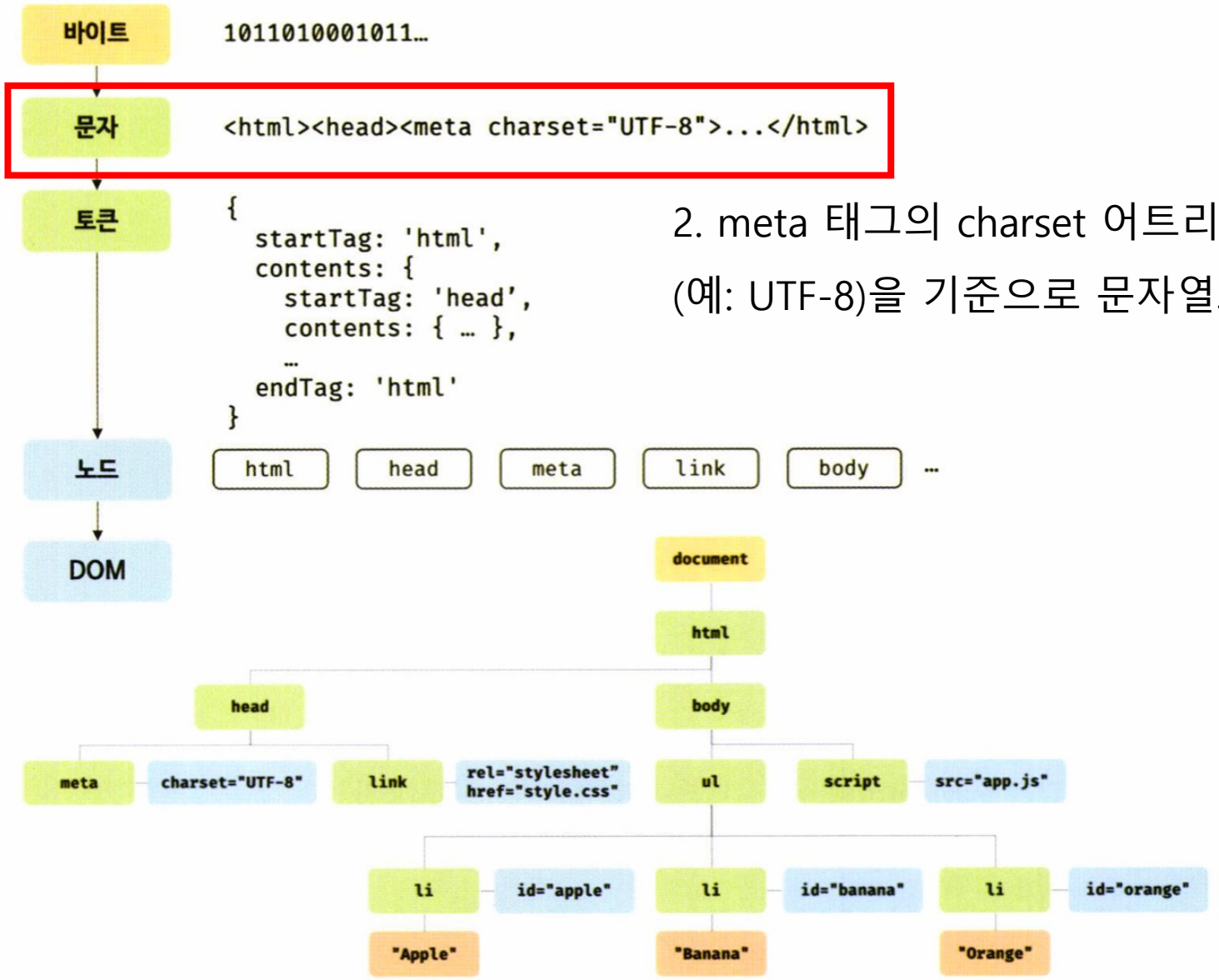


그림 38-6 HTML 파싱과 DOM 생성

2. meta 태그의 charset 어트리뷰트에 저장된 인코딩 방식 (예: UTF-8)을 기준으로 문자열로 반환된다.

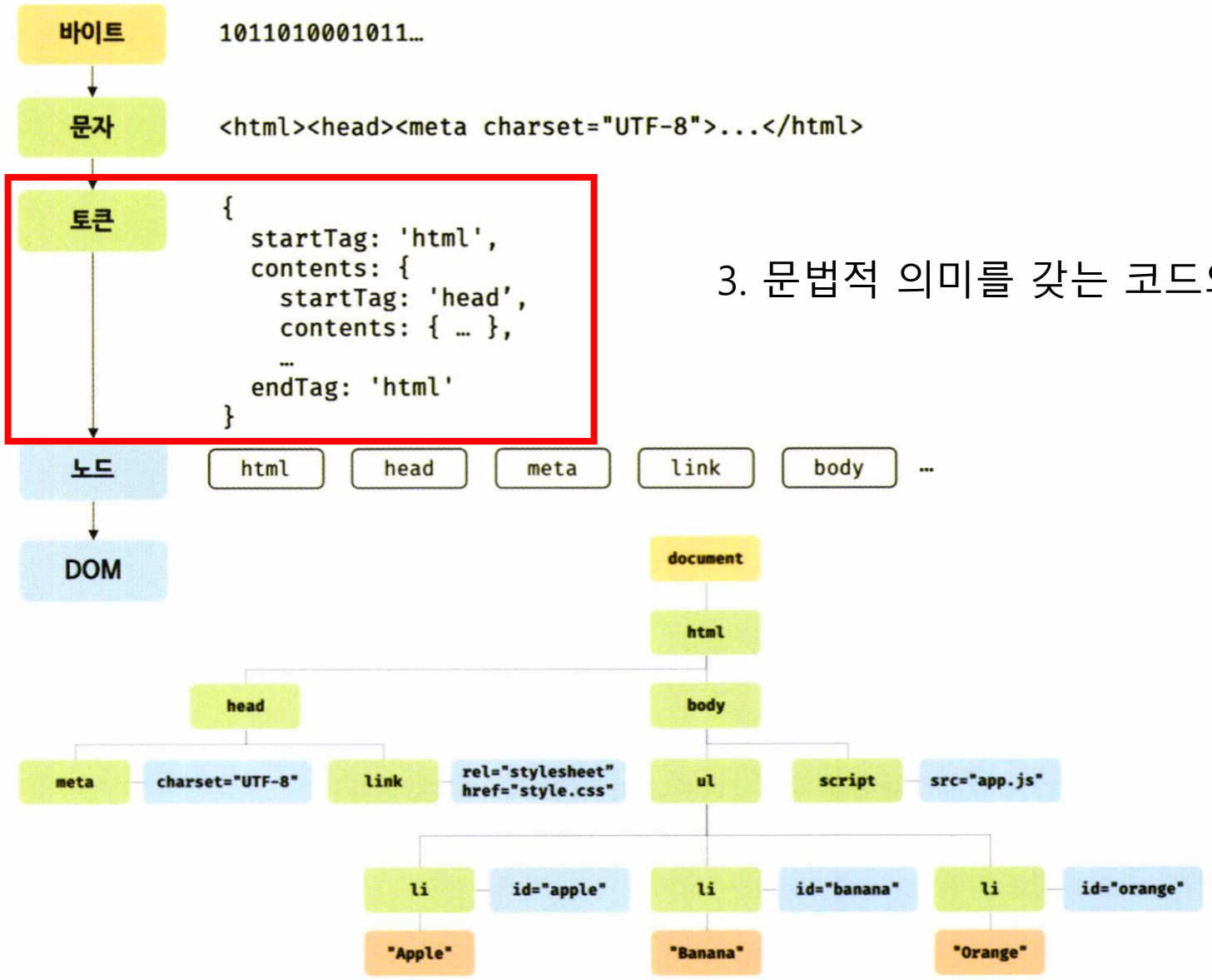


그림 38-6 HTML 파싱과 DOM 생성

3. 문법적 의미를 갖는 코드의 최소 단위 토큰들로 분해한다.

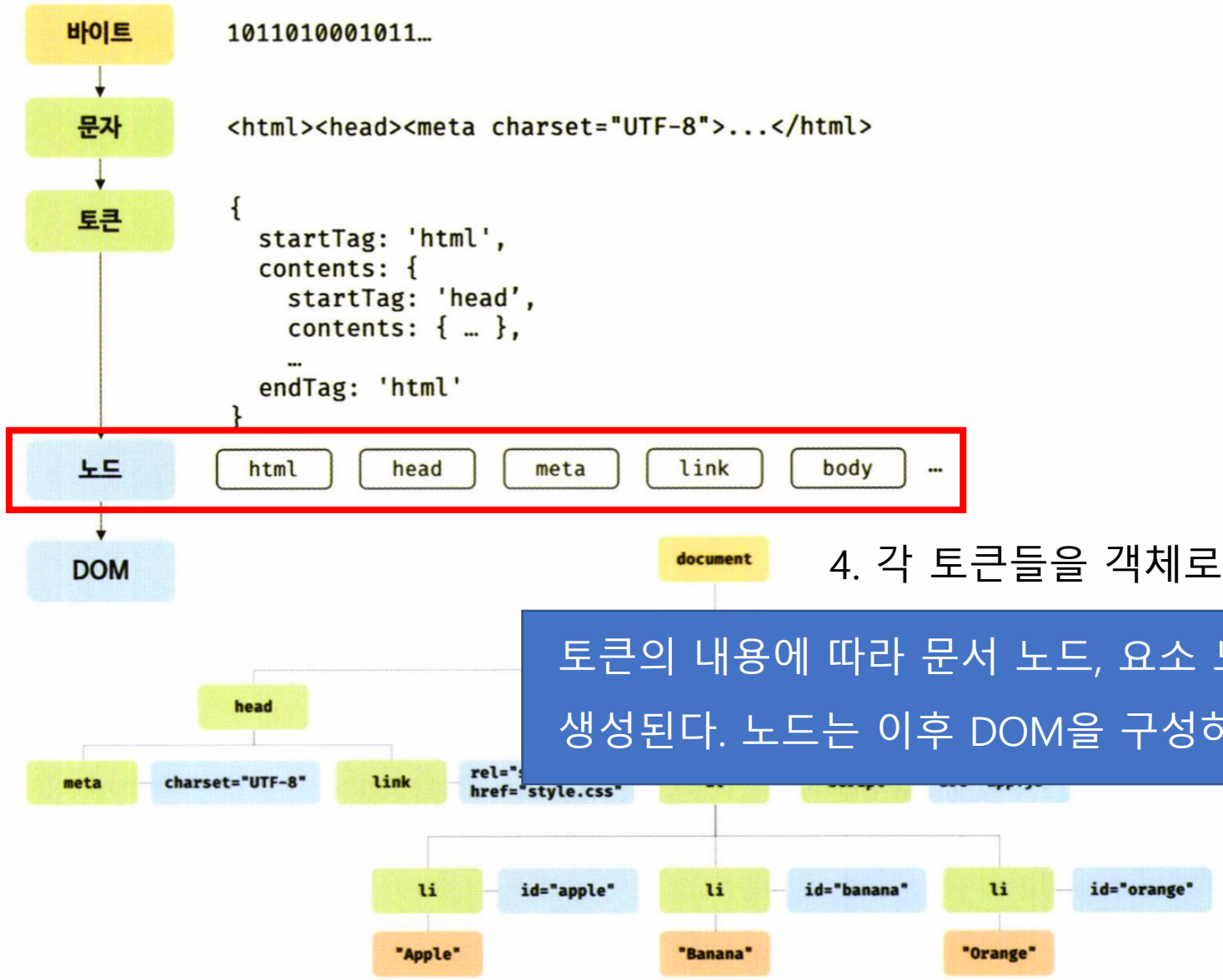


그림 38-6 HTML 파싱과 DOM 생성

4. 각 토큰들을 객체로 변환하여 노드들을 생성한다.

토큰의 내용에 따라 문서 노드, 요소 노드, 어트리뷰트 노드, 텍스트 노드가 생성된다. 노드는 이후 DOM을 구성하는 기본 요소가 된다.

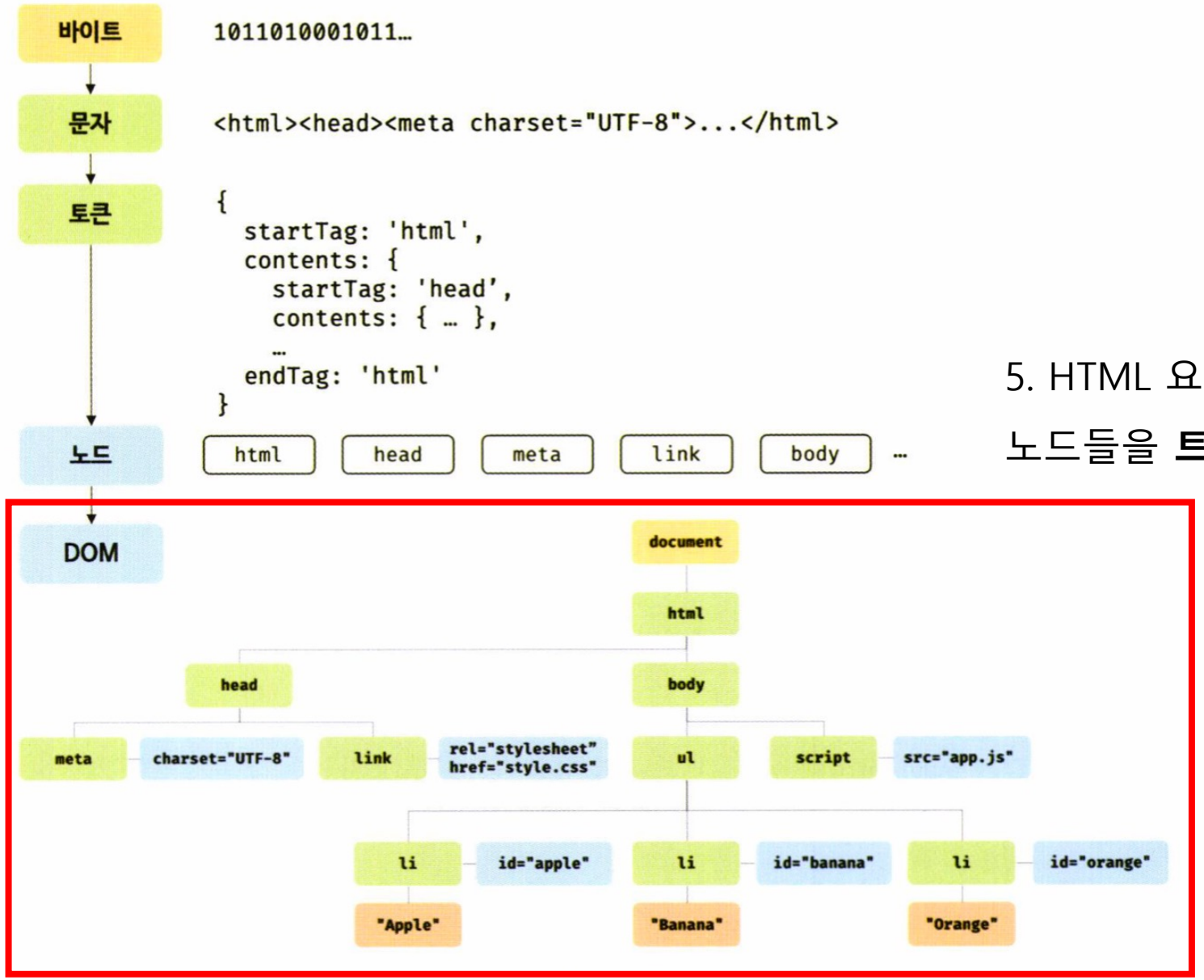


그림 38-6 HTML 파싱과 DOM 생성

5. HTML 요소 간 부자 관계를 반영하여 모든 노드들을 **트리 자료구조(DOM)**로 구성한다.

CSS 파싱과 CSSOM 생성

• 렌더링 엔진은 HTML을 처음부터 한 줄씩 순차적으로 파싱하여 DOM을 생성하다가 **link 태그나 style 태그를 만나면 DOM 생성을 일시 중단**

• CSS 파일을 로드하는 link 태그의 href 어트리뷰트에 지정된 CSS 파일을 서버에 요청하여 로드

• 로드한 CSS 파일이나 style 태그 내의 CSS를 HTML과 통일한 파싱 과정(바이트 → 문자 → 토큰 → 노드 → CSSOM) 과정을 거치며 해석하여 **CSSOM(CSS Object Model)**을 생성

• CSS 파싱을 완료하면 HTML 파싱이 중단된 지점부터 다시 HTML을 파싱하기 시작하여 DOM 생성을 재개

CSS 파싱과 CSSOM 생성

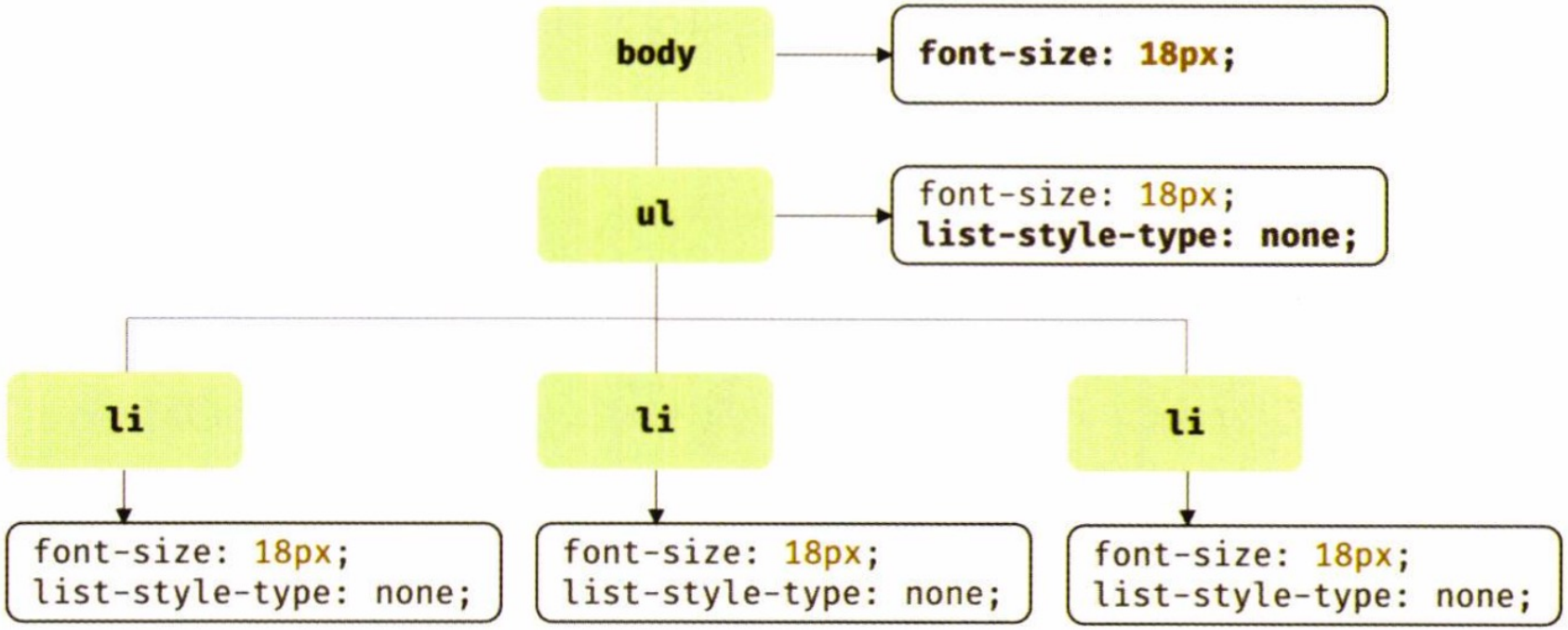


그림 38-7 CSSOM 생성

렌더트리 생성

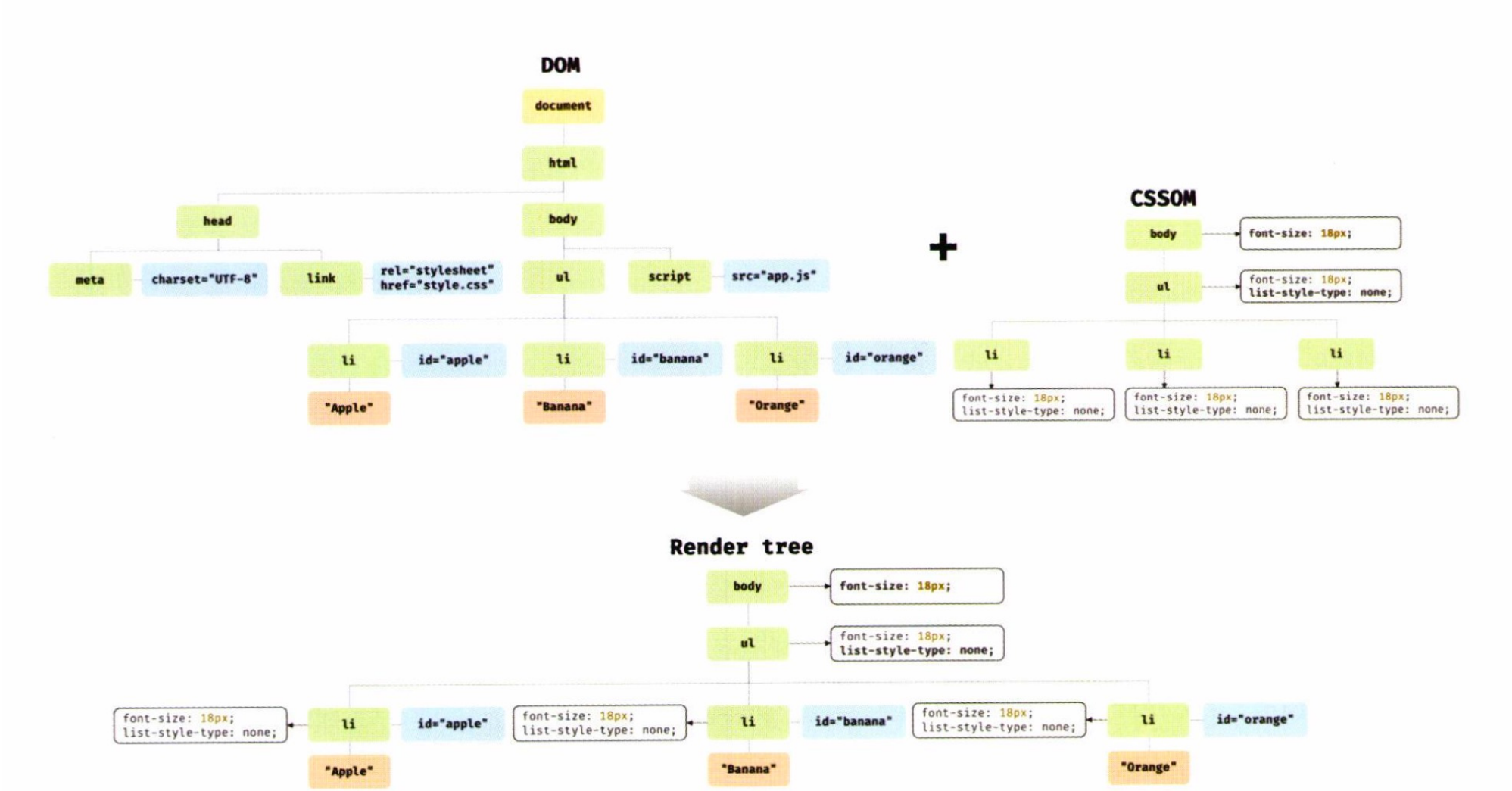


그림 38-8 렌더 트리 생성

브라우저 화면에 렌더링 되지 않는 노드(예: meta 태그, script 태그 등)와 CSS에 의해 비표시(예: display: none; 등)되는 노드들은 포함하지 않는다.

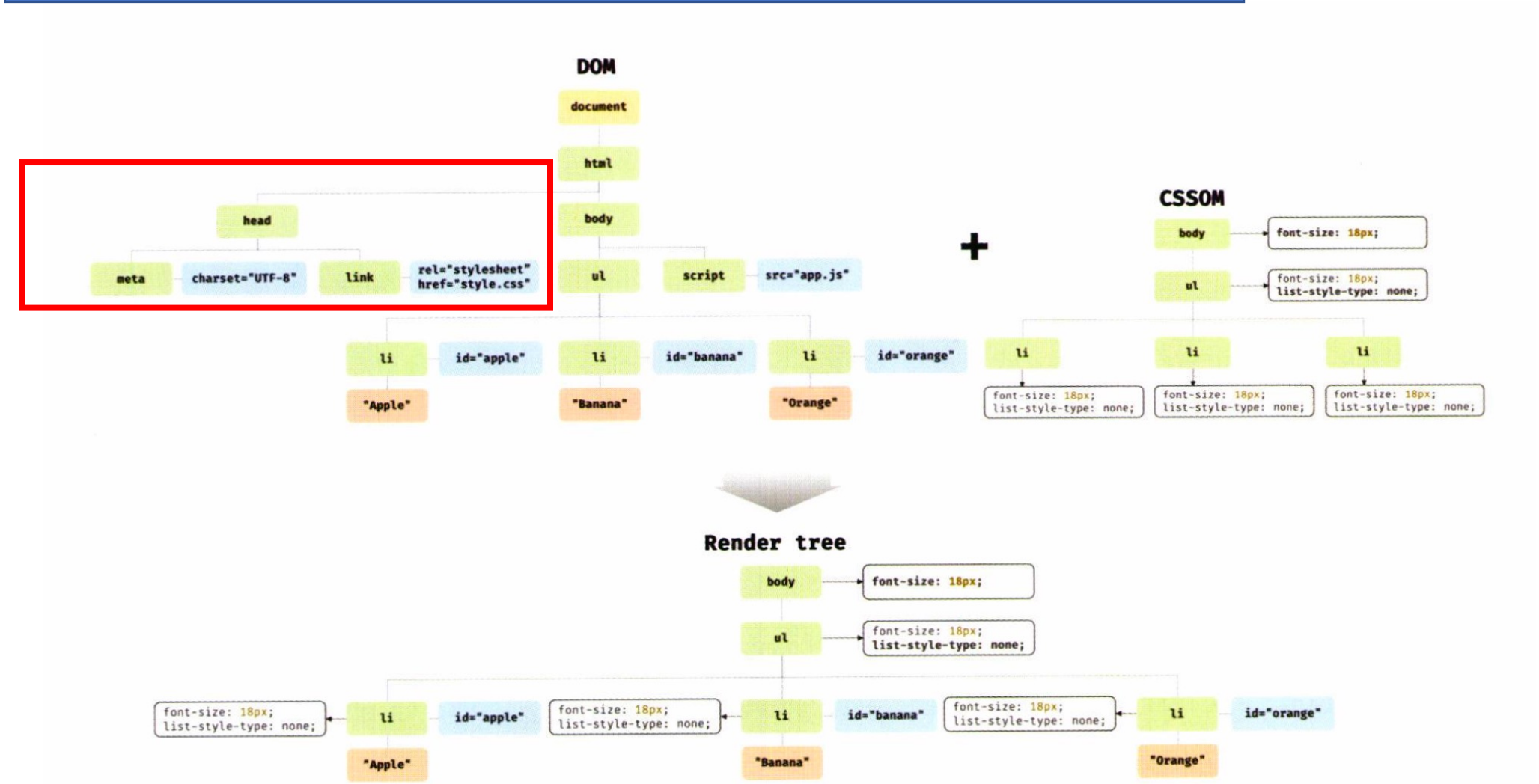


그림 38-8 렌더 트리 생성

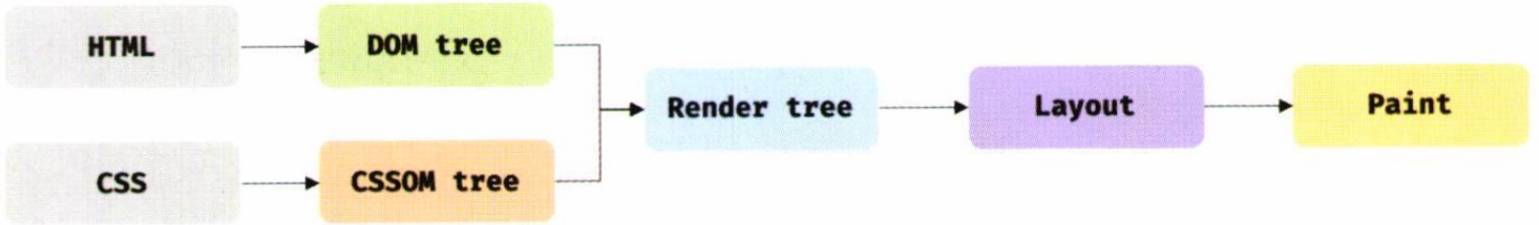
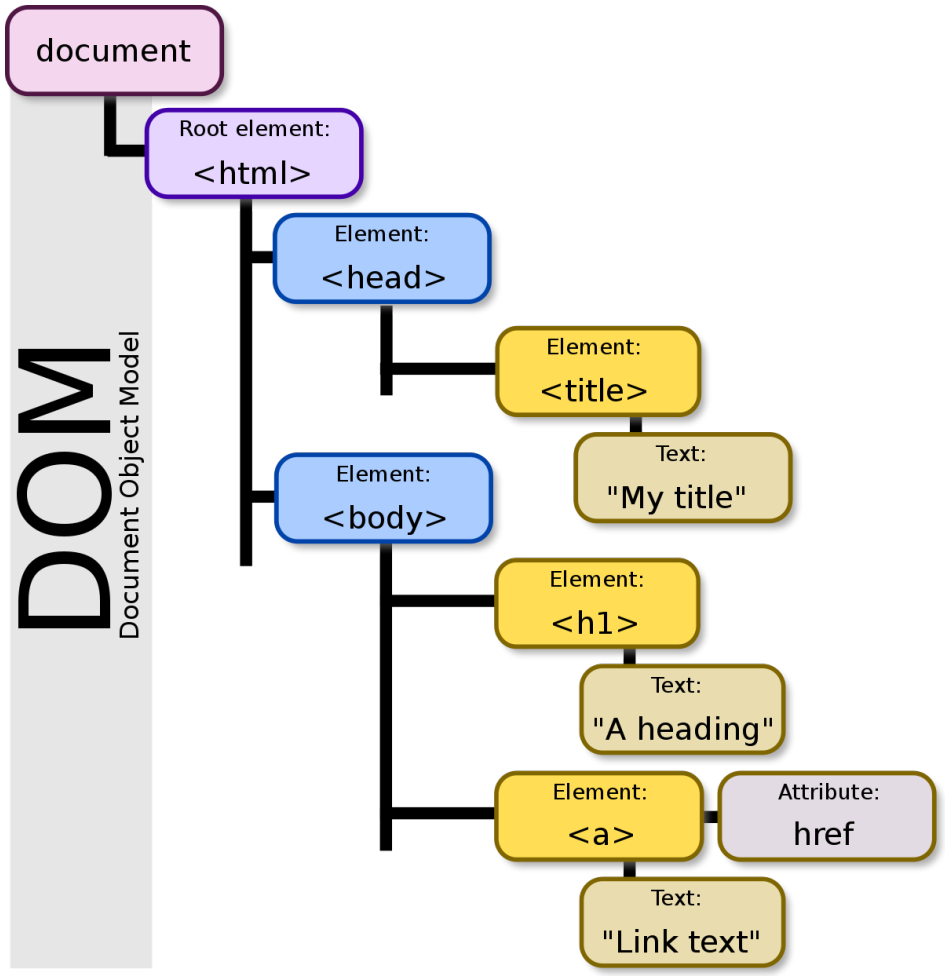


그림 38-9 렌더 트리와 레이아웃/페인트

- 완성된 렌더 트리는 각 HTML 요소의 레이아웃(위치와 크기)을 계산하는 데 사용되며 브라우저 화면에 픽셀을 렌더링 하는 **페인팅(painting)처리**에 입력된다.
- 다음과 같은 경우 해서 레이아웃 계산과 페인팅이 재차 실행된다.
  - 자바스크립트에 의한 노드 추가 또는 삭제
  - 브라우저 창의 리사이징에 의한 뷰포트(viewport) 크기 변경
  - HTML 요소의 레이아웃(위치, 크기)에 변경을 발생시키는 width/height, margin, padding, border, display, position, top/right/bottom/left 스타일 변경

자바스크립트 파싱과 실행



• DOM은 HTML 요소와 스타일 등을 변경할 수 있는 프로그래밍 인터페이스로서 **DOM API**를 제공

→ 자바스크립트 코드에서 DOM API를 통해 이미 생성된 DOM을 동적으로 조작 가능

자바스크립트 파싱과 실행

- 렌더링 엔진은 DOM을 생성하다가 script 태그를 만나면 **DOM 생성을 일시 중단**
    - 자바스크립트 파일을 로드하는 script 태그
    - 자바스크립트 코드를 콘텐츠로 담은 script 태그
- script 태그의 src 어트리뷰트에 정의된 자바스크립트 파일을 서버에 요청하여 로드한 자바스크립트 파일 또는 script 태그 내의 **자바스크립트 코드를 파싱하기 위해 자바스크립트 엔진에 제어권을 넘김**
- 이후 자바스크립트 파싱과 실행이 종료되면 렌더링 엔진으로 다시 제어권을 넘겨 HTML 파싱이 중단된 지점부터 다시 파싱을 시작하여 DOM 생성을 재개
- 자바스크립트 파싱과 실행은 렌더링 엔진이 아닌 자바스크립트 엔진이 처리

자바스크립트 엔진의 역할

- 자바스크립트 코드를 파싱하여 **CPU가 이해할 수 있는 저수준 언어로 변환하고 실행**
- 구글 크롬과 Node js의 V8, 파이어폭스의 SpiderMonkey, 사파리의 JavascriptCore 등 다양한 종류가 있고, **모든 자바스크립트 엔진은 ECMAScript 사양을 준수**

자바스크립트 엔진 파싱 과정

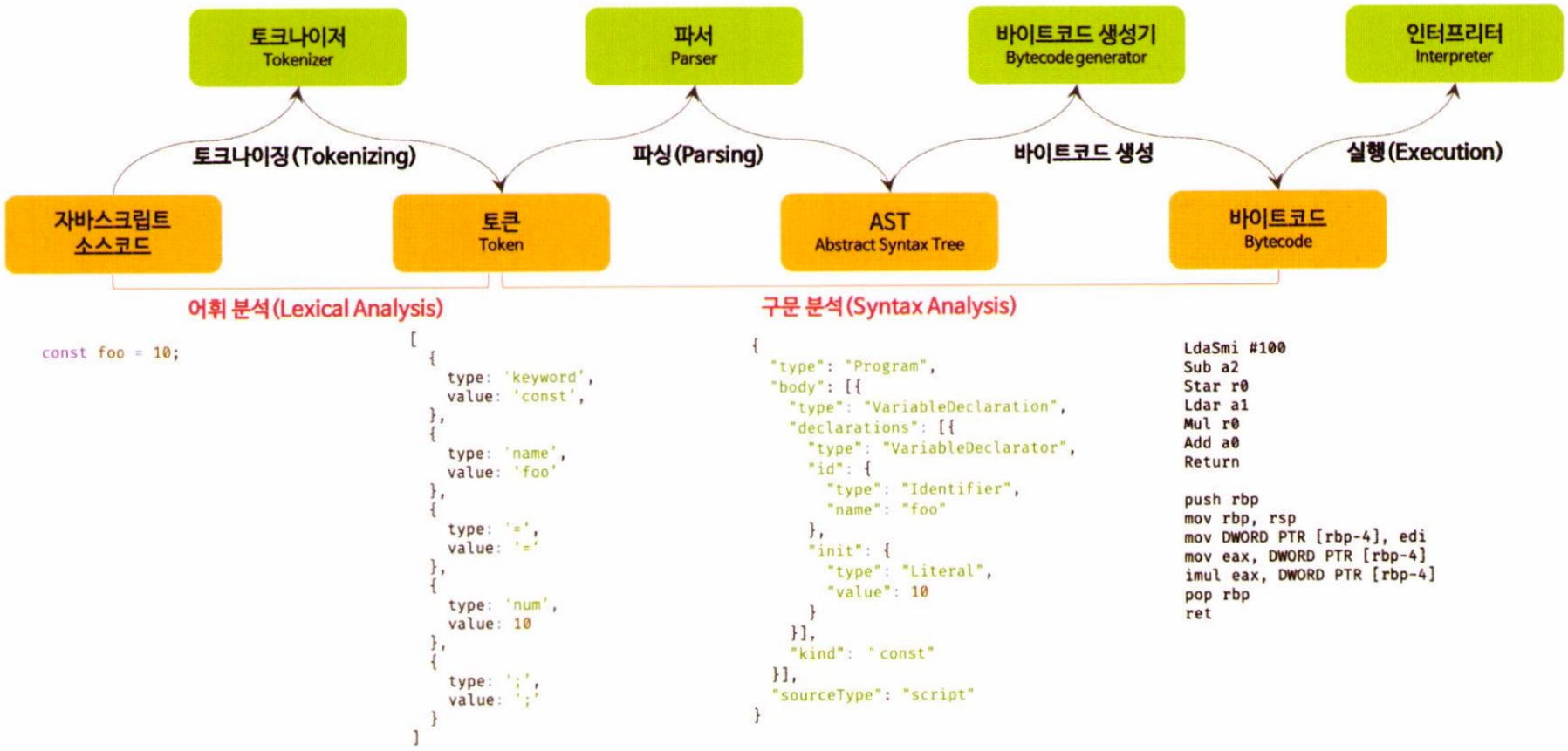
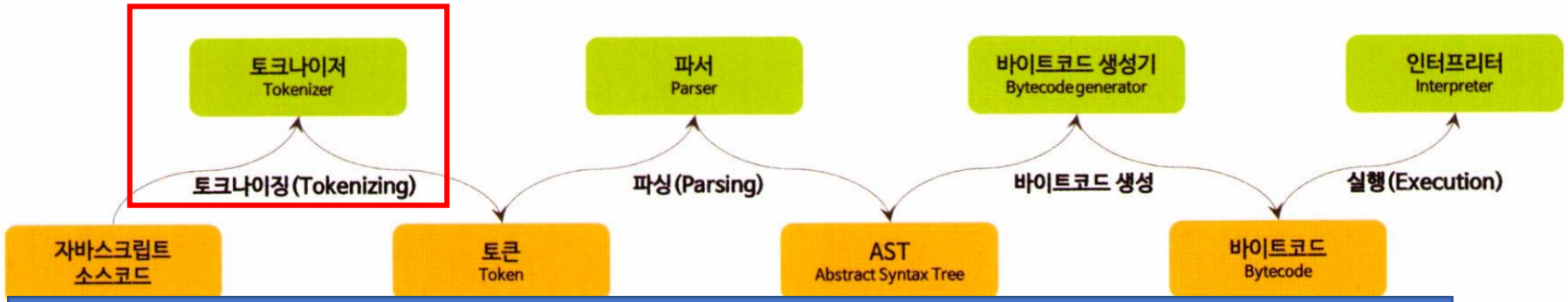


그림 38-10 자바스크립트 파싱과 실행

자바스크립트 엔진 파싱 과정



- 단순한 문자열인 자바스크립트 소스 코드를 어휘 분석(Exical analysis)하여 문법적 의미를 갖는 코드의 최소 단위인 토큰(token)들로 분해한다.

- 렉싱(lexing)이라고 부르기도 하지만 토크나이징과 미묘한 차이가 있음


```
{
  type: ';',
  value: ';'
}
]
```

```
  "kind": "const"
}],
"sourceType": "script"
}
```


그림 38-10 자바스크립트 파싱과 실행

자바스크립트 엔진 파싱 과정

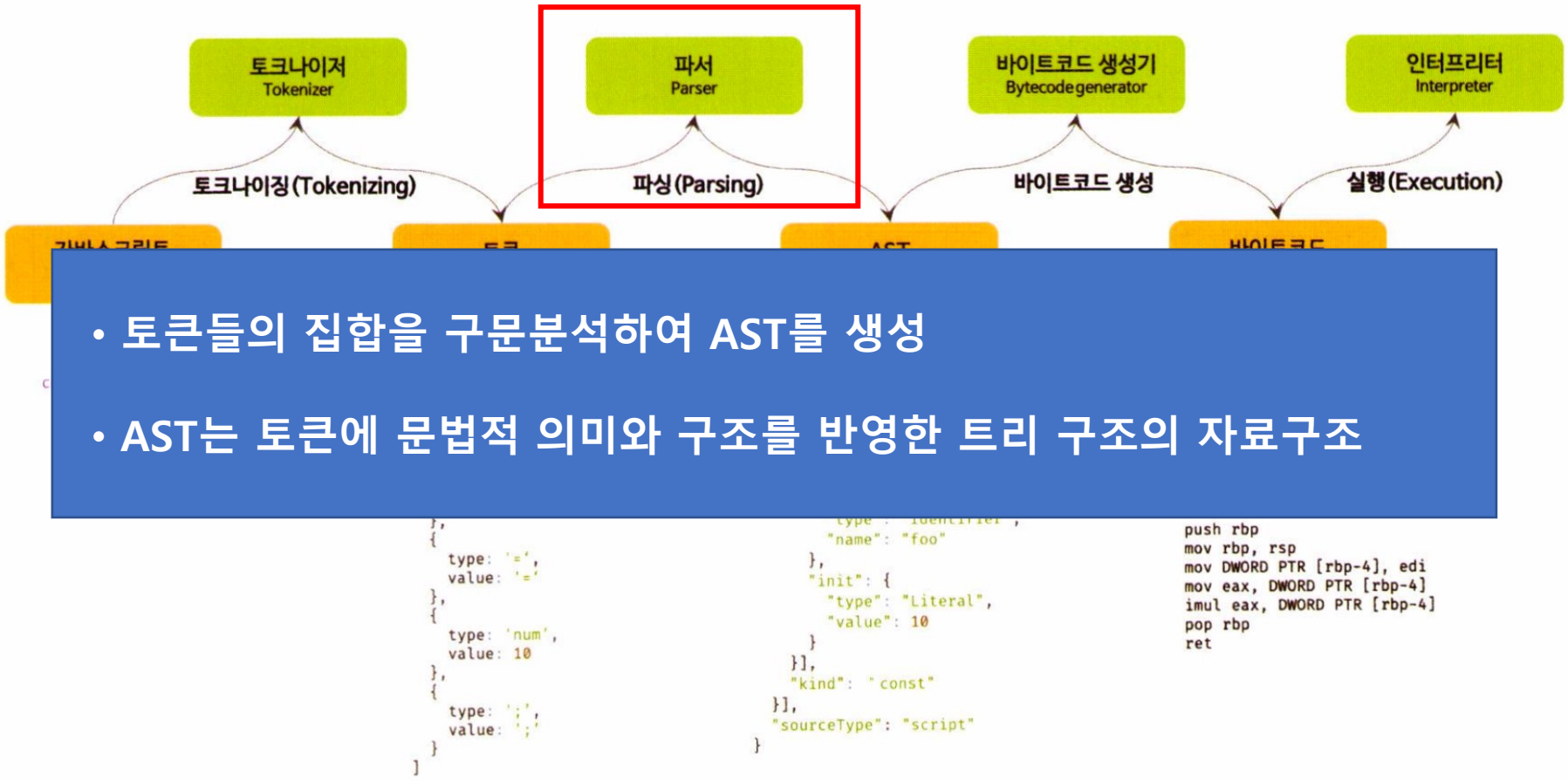


그림 38-10 자바스크립트 파싱과 실행

자바스크립트 엔진 파싱 과정

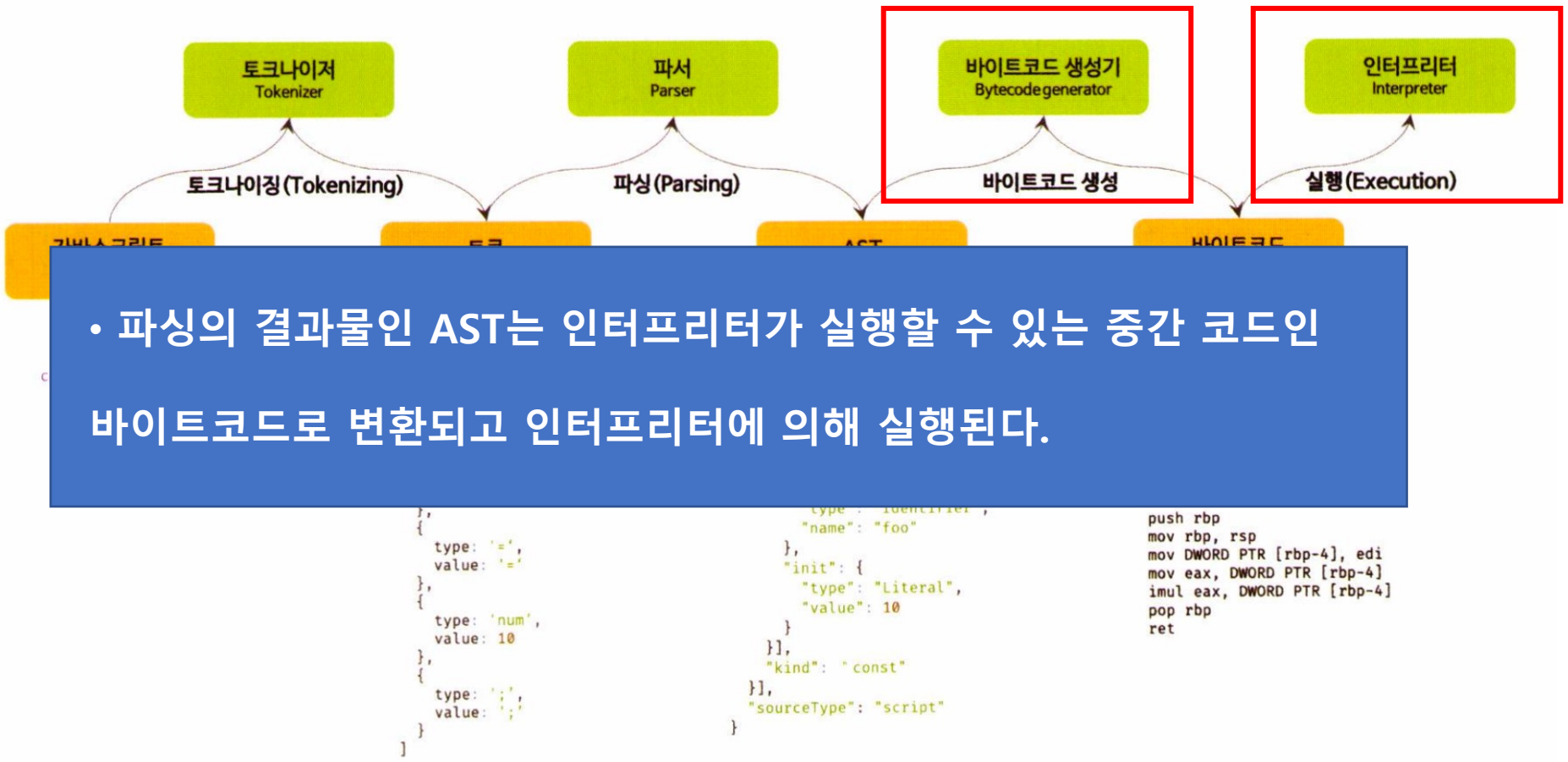


그림 38-10 자바스크립트 파싱과 실행

리플로우와 리페인트

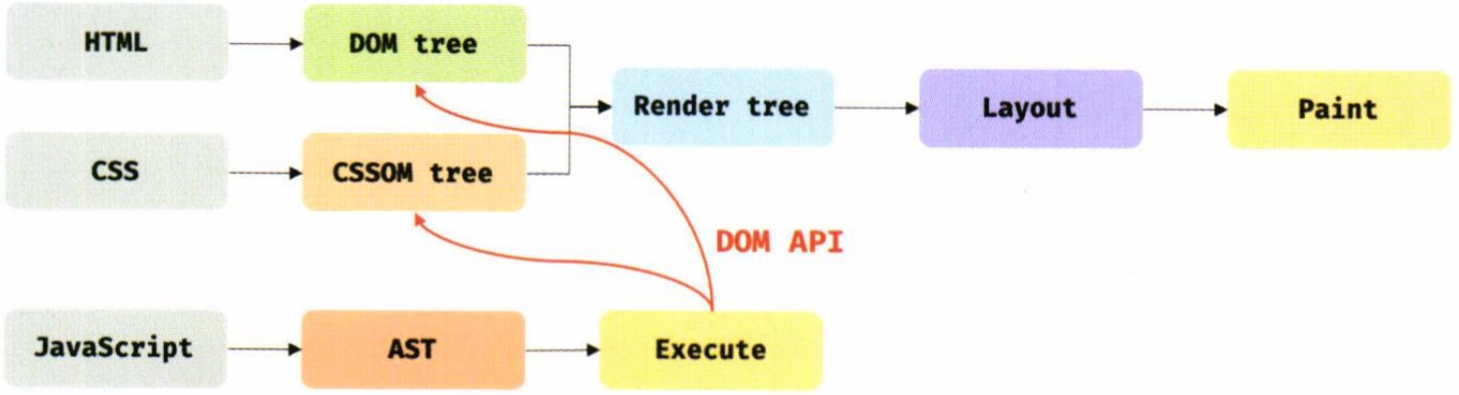


그림 38-12 DOM API에 의한 리플로우, 리페인트

- DOM이나 CSSOM을 변경하는 DOM API가 사용된 경우,

DOM이나 CSSOM은 변경되어 다시 렌더 트리로 결합된다.

- 변경된 렌더트리를 기반으로 레이아웃과 페인트 과정을 거쳐

브라우저 화면에 다시 렌더링한다. → **리플로우, 리페인트**

리플로우와 리페인트

- 리플로우는 레이아웃 계산을 다시 하는 것을 말하며, 노드 추가/삭제, 요소의 크기/위치 변경, 윈도우 리사이징 등 **레이아웃에 영향을 주는 변경이 발생한 경우에 한하여 실행된다.**
- 리페인트는 **재결합된 렌더트리를 기반으로 다시 페인트**를 하는 것을 말한다.
- **만약 레이아웃에 영향이 없는 변경은 리플로우 없이 리페인트만 실행된다**.

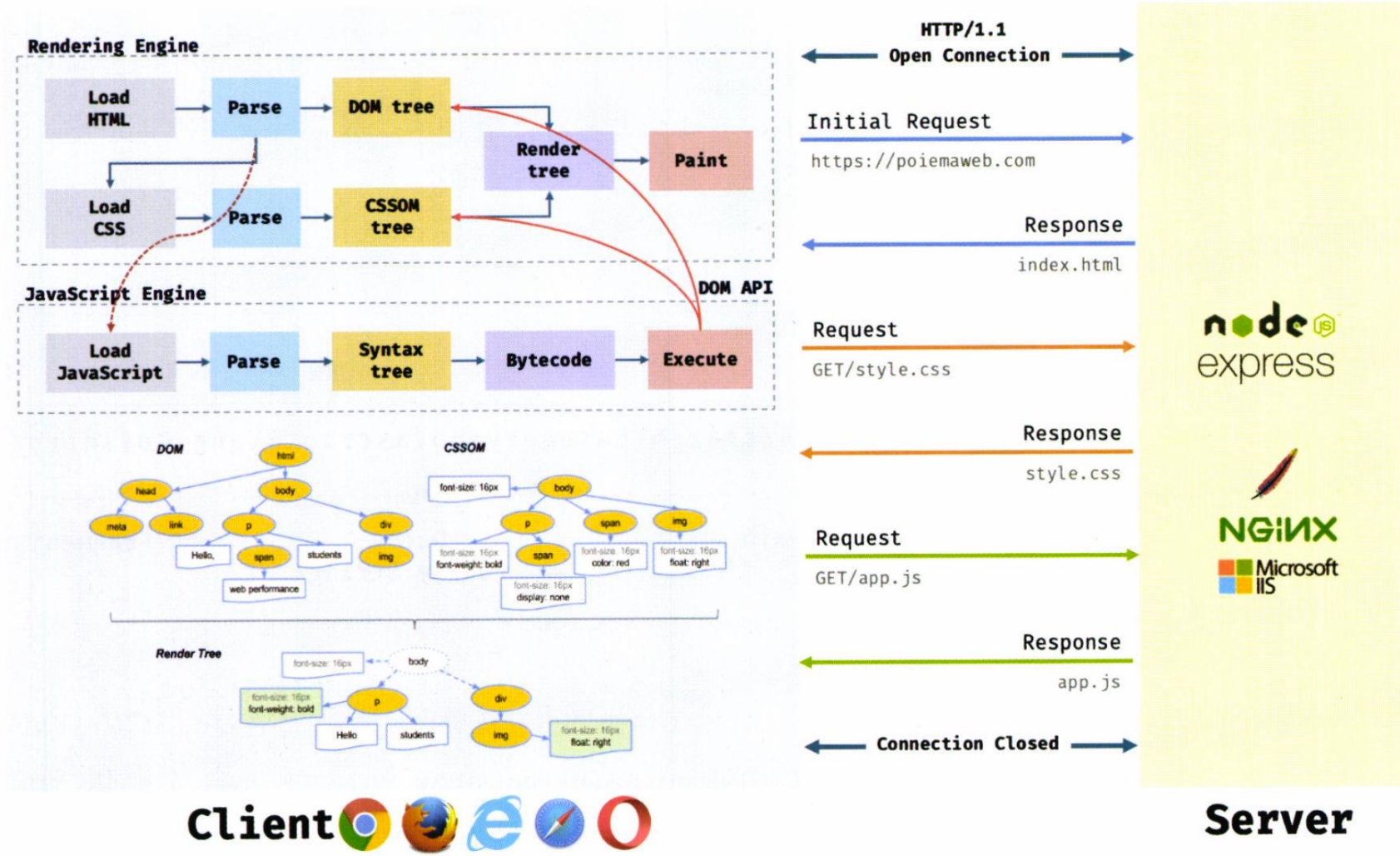


그림 38-1 브라우저의 렌더링 과정