

# CORS



여민수



# CORS

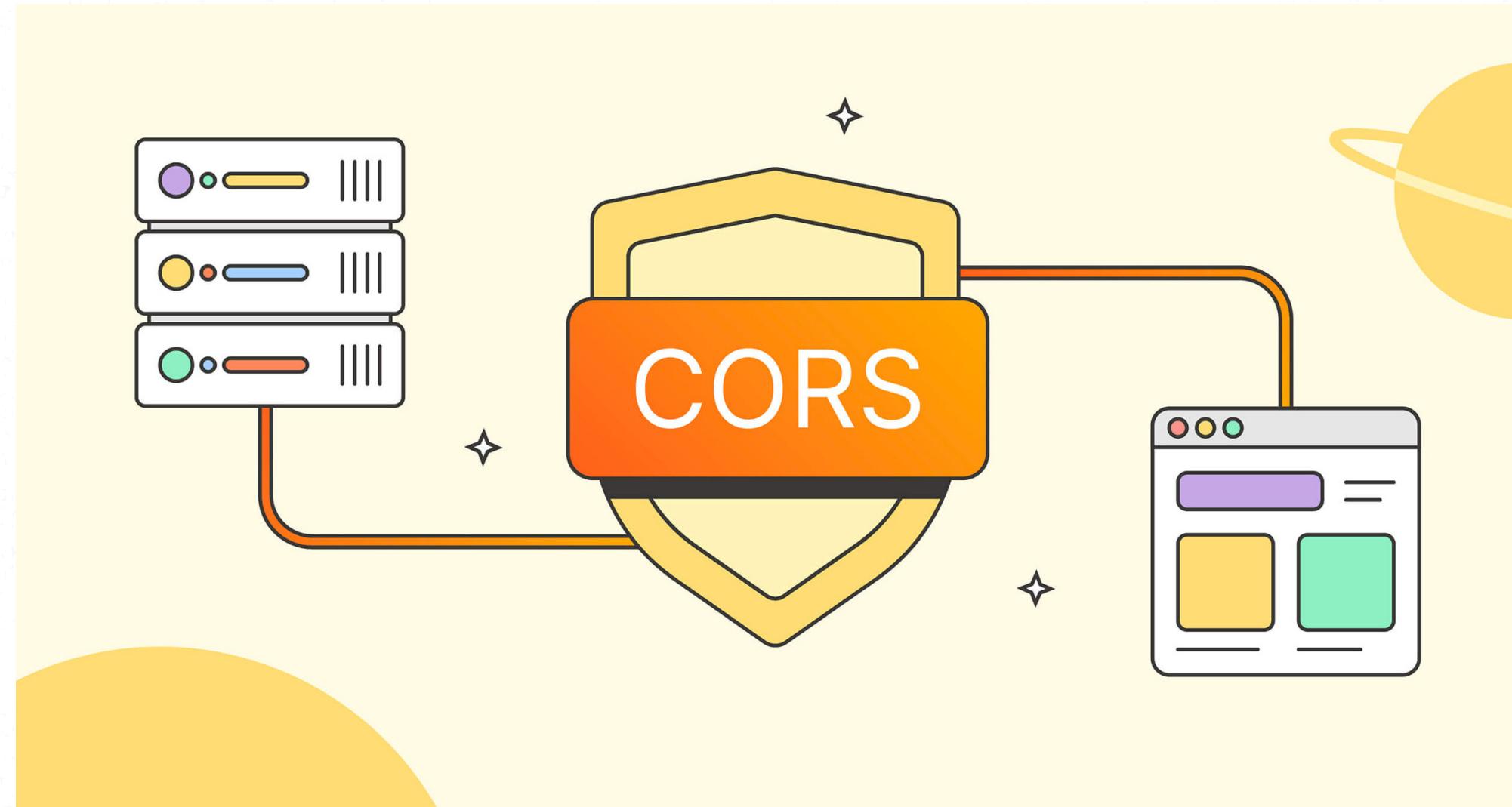
Cross-origin-resource-sharing

```
Access to XMLHttpRequest at 'https://localhost:44300/api/route/'  
from origin 'http://localhost:3000' has been blocked by CORS policy:  
Response to preflight request doesn't pass access control check: No  
'Access-Control-Allow-Origin' header is present on the requested  
resource.
```

웹 개발을 하다보면 반드시 한 번쯤은 CORS 에러가 발생!

분명히 POSTMAN이나 백엔드에서  
HTTP 요청으로 보내면 잘 되는데...

## □ CORS란?

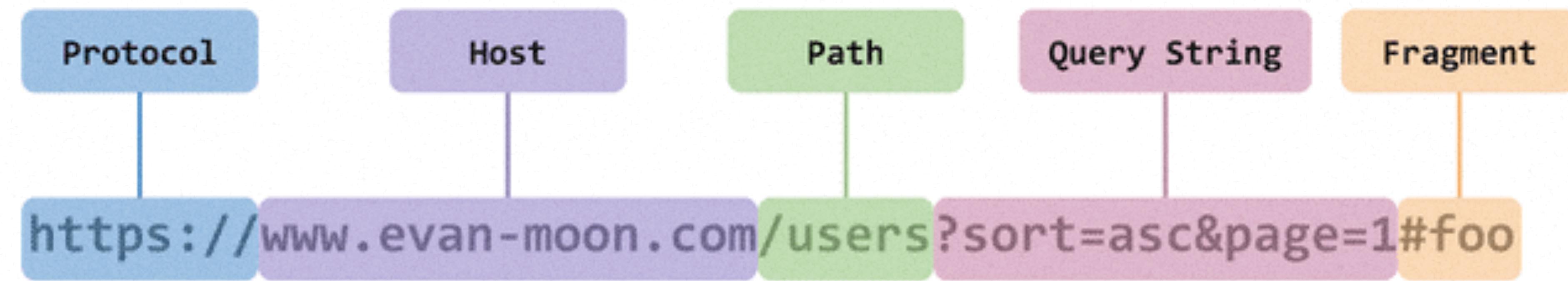


CORS는 Cross-Origin Resource Sharing의 약자

직역하면 **다른 출처 리소스 공유**

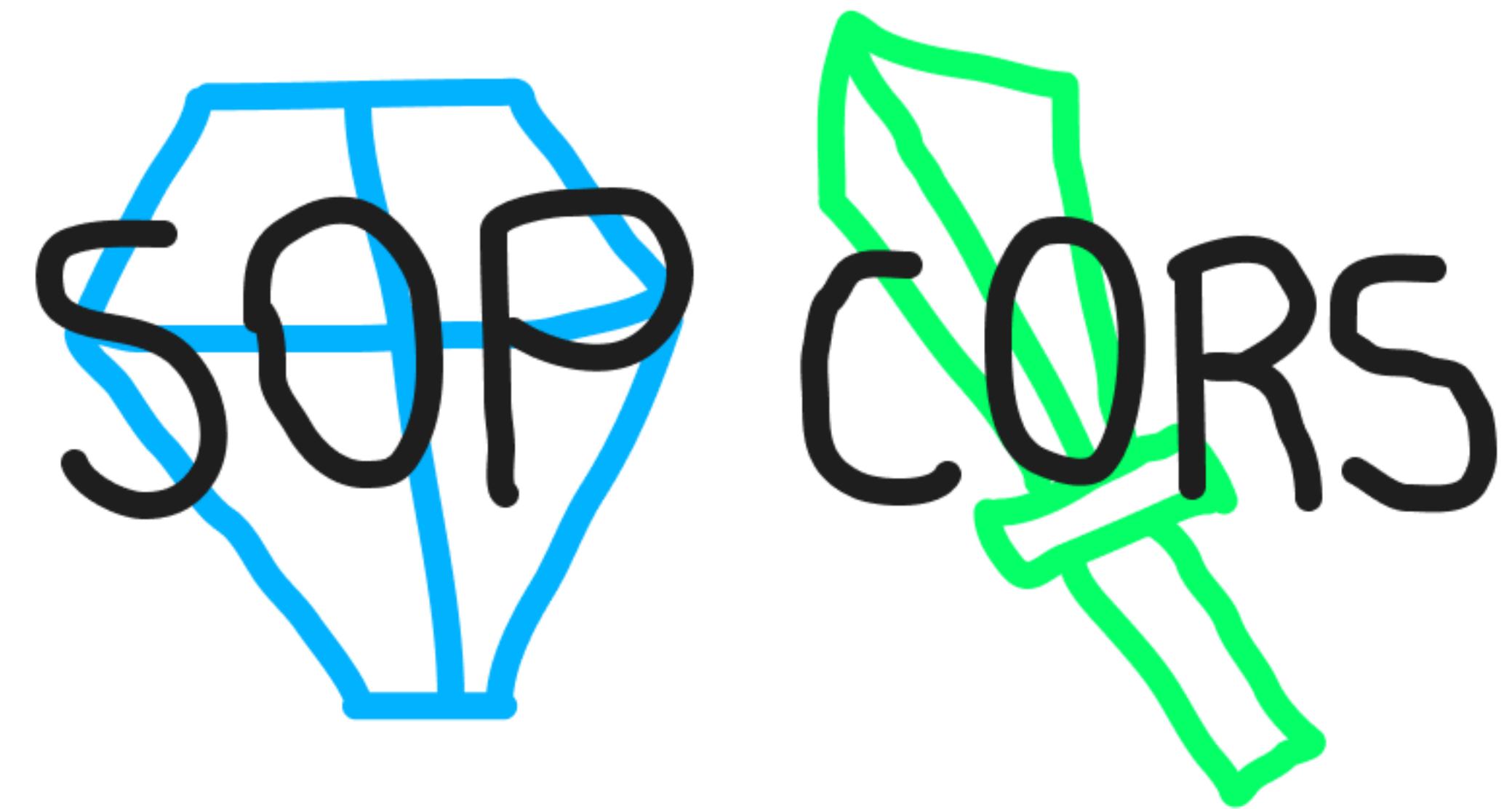
**즉, “다른 출처” 간 리소스 공유 과정에서 에러가 발생했다는 의미**

## □ 출처가 뭔데?



출처(Origin): Protocol과 Host, 그리고 포트번호를 합친 것을 의미

## □ 웹 브라우저의 두가지 정책

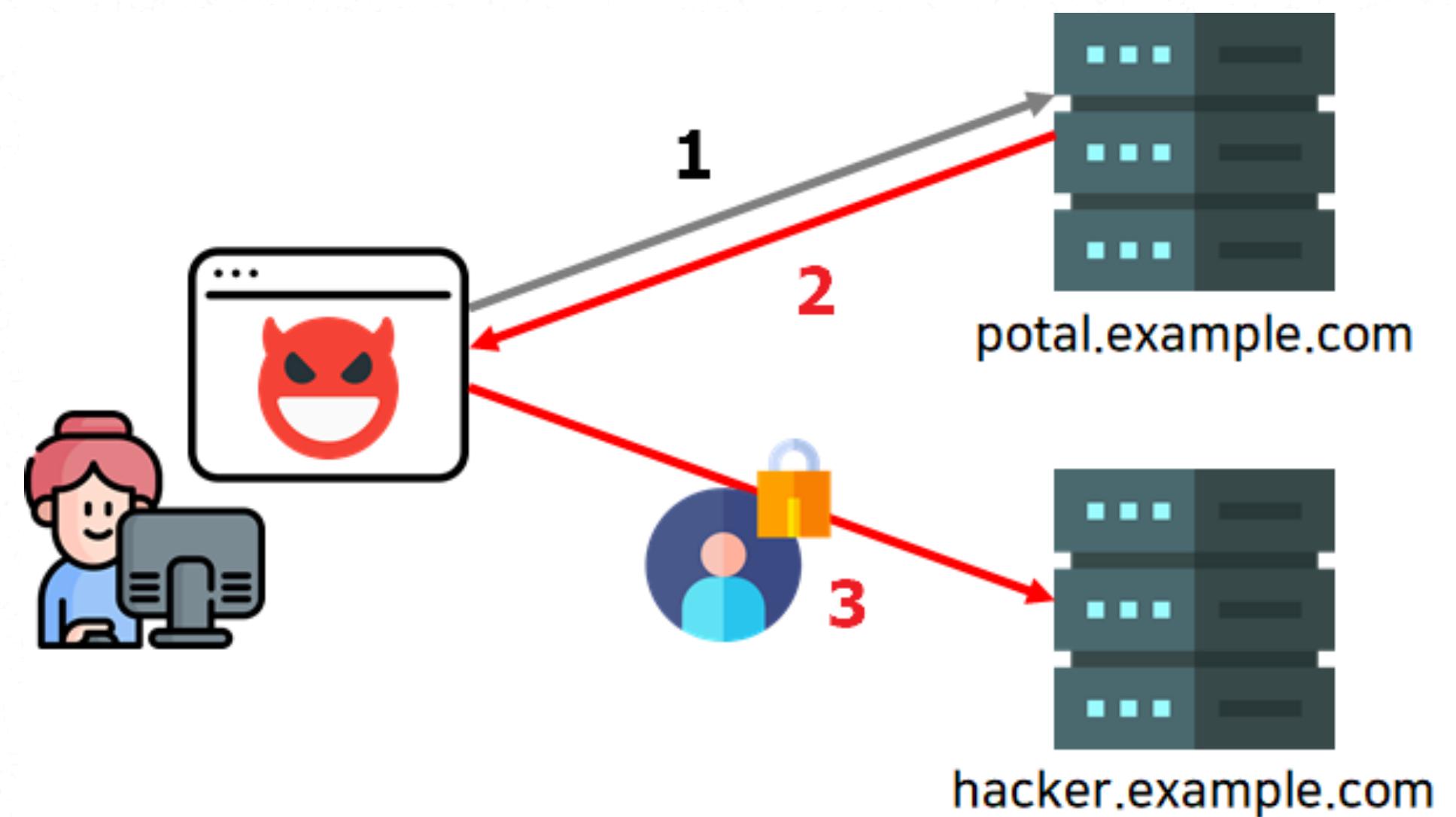


웹 세계에서는 다른 출처로의 리소스 요청을 제한하는 것과 관련된 두가지 정책 존재

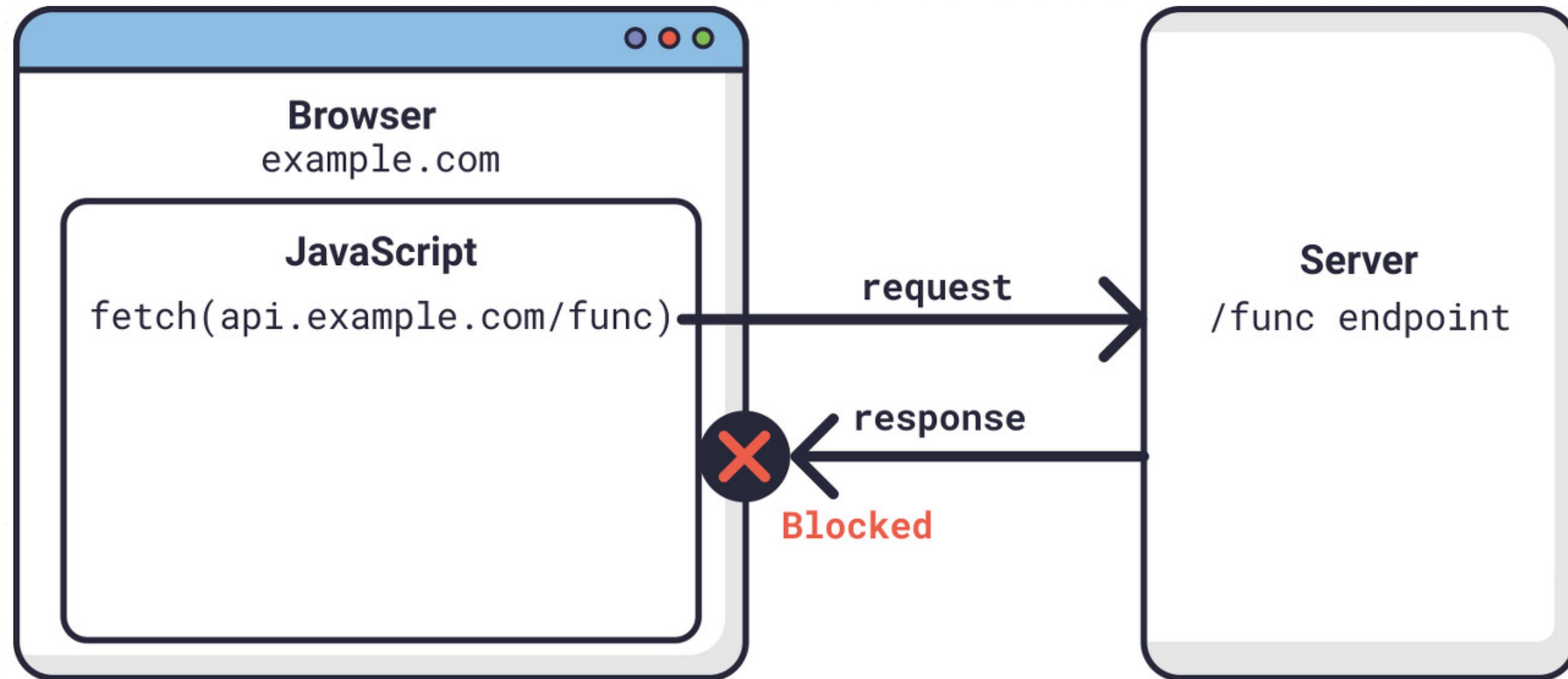
## SOP(Same-Origin-Policy)

2011년 처음 등장한 보안 정책으로, 같은 출처에서만 리소스를 공유할 수 있다는 규칙

### 규칙이 나타난 이유



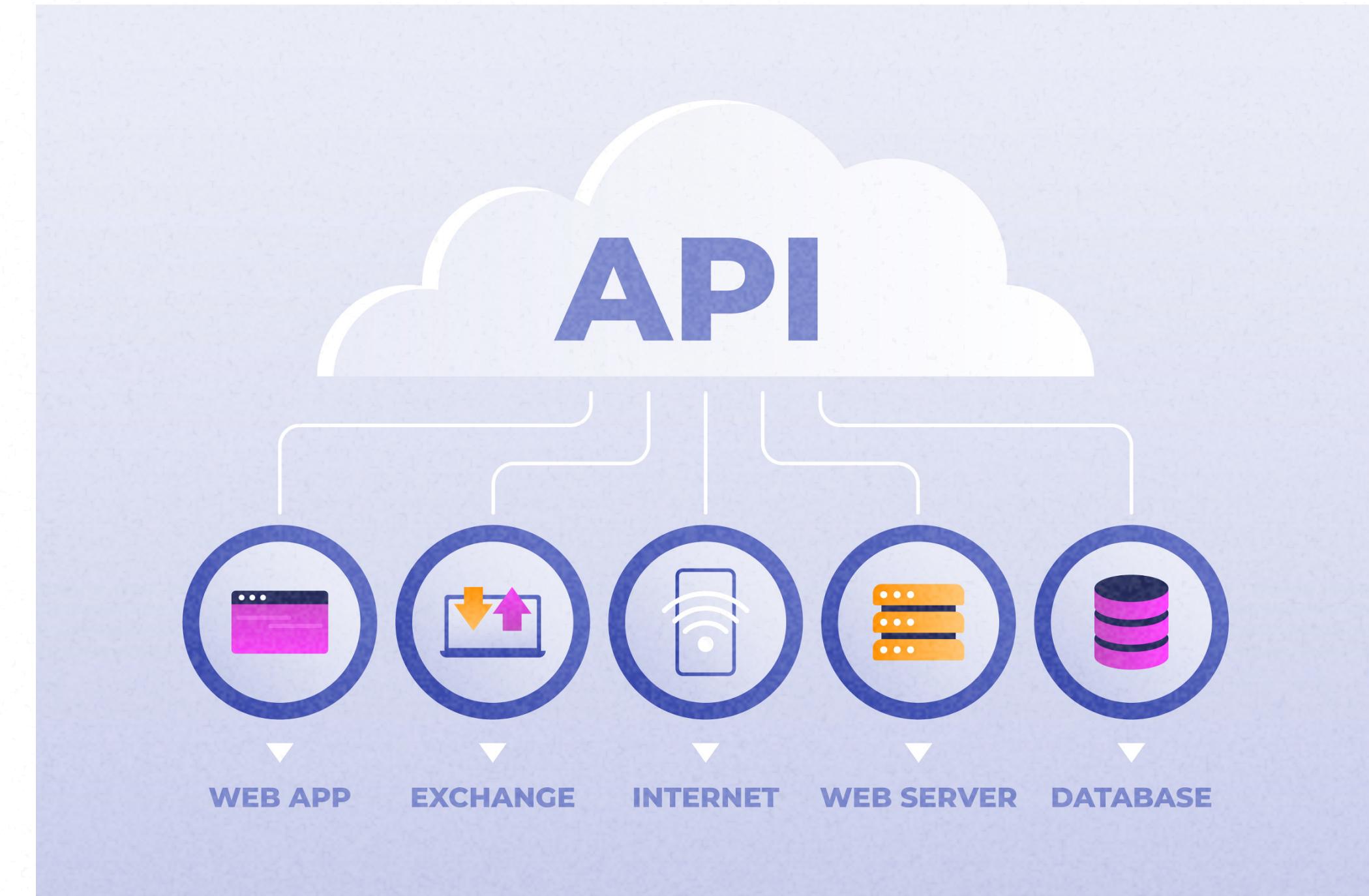
## □ 출처비교와 차단



**출처비교는 서버가 하는 것이 아니라 브라우저가 판단한다!**

(POSTMAN에서 되던게 웹 브라우저에선 안되는 이유)

## □ 다른 출처에서 리소스를 가져오려면?

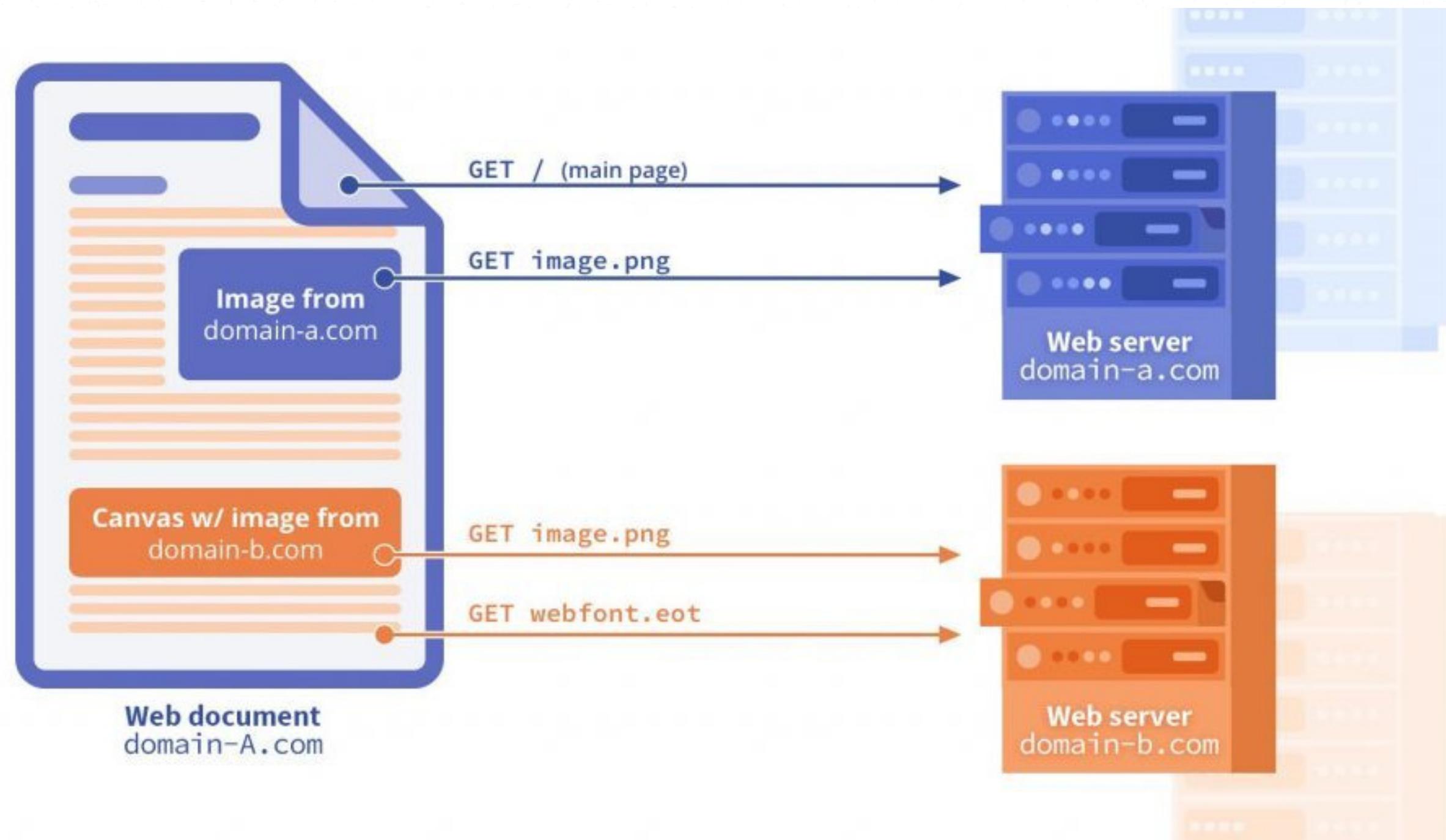


웹 개발을 하다보면 보안도 보안이지만 다른 출처 간 상호작용을 해야하는 케이스가 반드시 존재!

ex) 네이버 지도 API와 같은 OPEN API 사용

이러한 예외사항 대해 제한적으로 **CORS 정책을 허용하는 리소스**에 대해선 다른 출처라도 받아드리기로 함!

## □ 사실 CORS는 해결책이였다?



웹 브라우저에서 보이던 새빨간 에러메시지는 **SOP정책에 따라 다른 출처의 리소스를 차단하면서 발생된 에러!**

SOP에 막히더라도 **CORS 정책에 따르면 다른 출처의 리소스라도 가져올 수 있다!**

## □ CORS 동작 방식

### 1. 클라이언트에서 HTTP요청의 헤더에 Origin을 담아 전달

요청 헤더    소스 보기

```
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Access-Control-Request-Headers: content-type
Access-Control-Request-Method: POST
Connection: keep-alive
Host: localhost:4000
Origin: http://localhost:3000
Referer: http://localhost:3000/
```

### 2. 서버는 응답헤더에 Access-Control-Allow-Origin을 담아 클라이언트로 전달한다.

▶ 응답 헤더    소스 보기

```
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: content-type
Access-Control-Allow-Methods: GET,POST,OPTIONS,DELETE,PUT,PATCH
Access-Control-Allow-Origin: http://localhost:3000
Connection: keep-alive
```

### 3. 클라이언트에서 Origin과 서버가 보내준 Access-Control-Allow-Origin을 비교한다.

## □ CORS 동작 3가지 시나리오



### 예비 요청 (Preflight Request)

요청을 보낼때 예비 요청을 보내 서버와 잘 통신되는지 확인한 후 본 요청을 보냄

### 단순 요청(Simple Request)

예비 요청(Preflight)을 생략하고 바로 서버에 직행으로 본 요청을 보냄

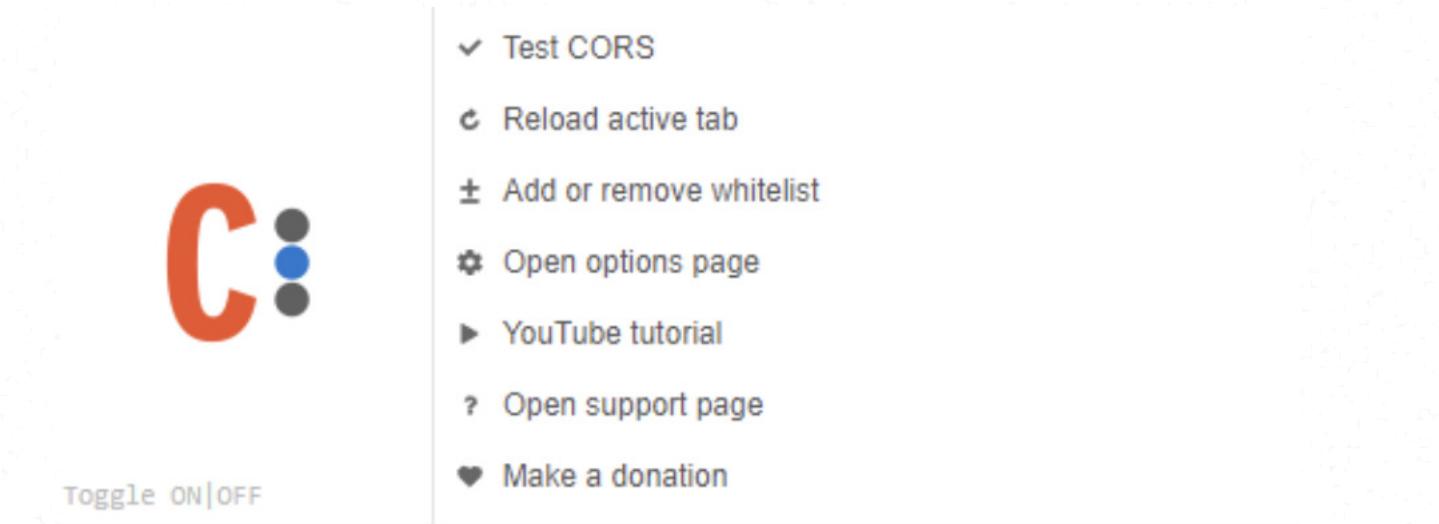
### 인증된 요청 (Credentialed Request)

클라이언트에서 서버에게 **자격 인증 정보(Credential)**를 실어 요청할때 사용되는 요청

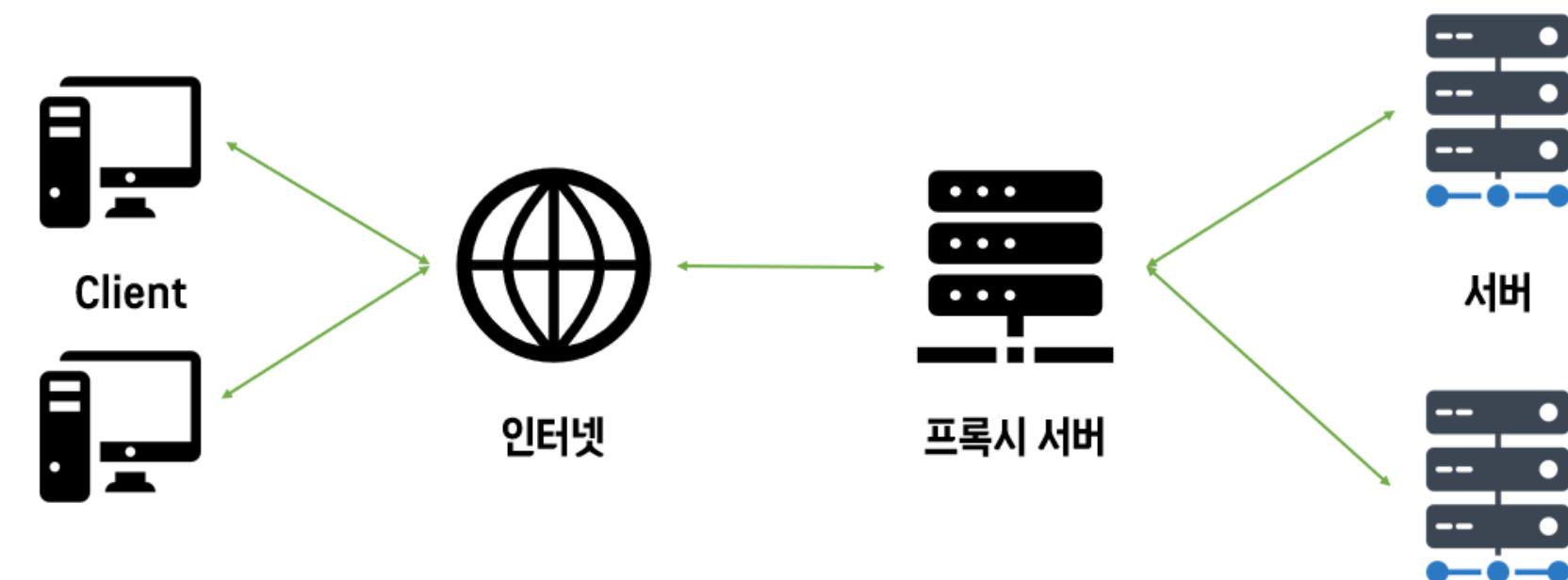
💡 **자격 인증 정보란?** 세션 ID가 저장되어있는 쿠키(Cookie) 혹은 Authorization 헤더에 설정하는 토큰 값 등

## □ 그래서 어떻게 CORS를 해결하는데?

### 1. Chrome 확장 프로그램 이용 (Allow CORS: Access-Control-Allow-Origin)



### 2. 프록시 사이트 이용하기



### 3. 서버에서 Access-Control-Allow-Origin 헤더 세팅하기

```

JAVA
1 // 스프링 서버 전역적으로 CORS 설정
2 @Configuration
3 public class WebConfig implements WebMvcConfigurer {
4     @Override
5     public void addCorsMappings(CorsRegistry registry) {
6         registry.addMapping("/**")
7             .allowedOrigins("http://localhost:8080", "http://localhost:8081") // 허용할 출처
8             .allowedMethods("GET", "POST") // 허용할 HTTP method
9             .allowCredentials(true) // 쿠키 인증 요청 허용
10            .maxAge(3000) // 원하는 시간만큼 pre-flight 리퀘스트를 캐싱
11     }
12 }
  
```