HTTP, async/await

8 / HTTP, async/await

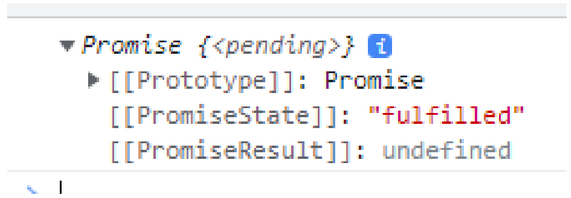
목차

01

async, await

async, await

```
console.log
(
new Promise((resolve) => {
  resolve();
}).then()
);
```



- **Promise**
  - 비동기 작업을 처리하기 위한 객체이자 패턴
- **then**
  - 프로미스 후속 처리 메서드
  - then을 호출하면 Promise 객체를 반환
  - 프로미스의 상태 변화시 then의 인수인 콜백 함수가 호출

async, await

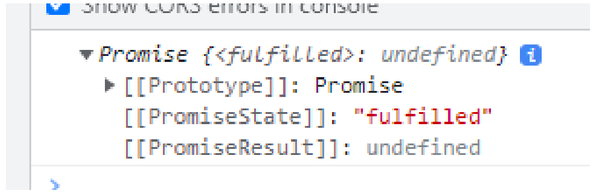
- 비동기 처리 패턴 중 하나로, 비동기 처리를 동기 처리처럼 동작하도록 구현할 수 있음
- ES8에서 나온 문법으로, 비동기 코드를 더 읽기 쉽게 작성하도록 도와줌
- async 키워드를 사용한 함수는 프로미스를 반환함
- await 키워드는 반드시 async 함수 내부에서 사용해야 함

async, await

- async, await 예제

```
async function foo(n) { }

console.log(foo())
```




```
async function foo(n) {
  return n;
}

foo(1).then(v => console.log(v)); // 1
```

- async 함수는 function 키워드 앞에 async를 붙여서 만듬
- async 함수는 언제나 프로미스를 반환
- 프로미스를 반환하니 then 메서드를 연달아 쓸 수 있음

async, await

- async, await 예제

```
function sleep() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('result')
    }, 1000)
  });
}

async function process() {
  console.log('비동기 시작');
  let result = await sleep();
  console.log('비동기 끝');
  console.log(result);
}

process();
```

- await 키워드는 프로미스가 settled 상태가 될 때까지 대기하다가 settled 상태가 되면 프로미스가 resolve한 결과를 처리함

  settled: 비동기 처리가 수행된 상태

- await 키워드는 반드시 프로미스 앞에서 사용해야 함

async, await

- async, await 연달아 쓰기

```
async function process() {
  const dog = await getDog();
  console.log(dog);
  const rabbit = await getRabbit();
  console.log(rabbit);
  const turtle = await getTurtle();
  console.log(turtle);
}
```

- await 키워드는 then 메서드 체인을 연결한 것처럼 순서대로 동작한다.
- 비동기 코드에 쉽게 순서를 부여한다

async, await

- try-catch

```
async function process () {
  try {
    await sleep(1000);
    const error = new Error();
    throw error;
  } catch (e) {
    console.error(e);
  }
}
```

```
async function process () {
  await sleep(1000);
  const error = new Error();
  throw error;
}

process()
.then(console.log)
.catch(console.error);
```

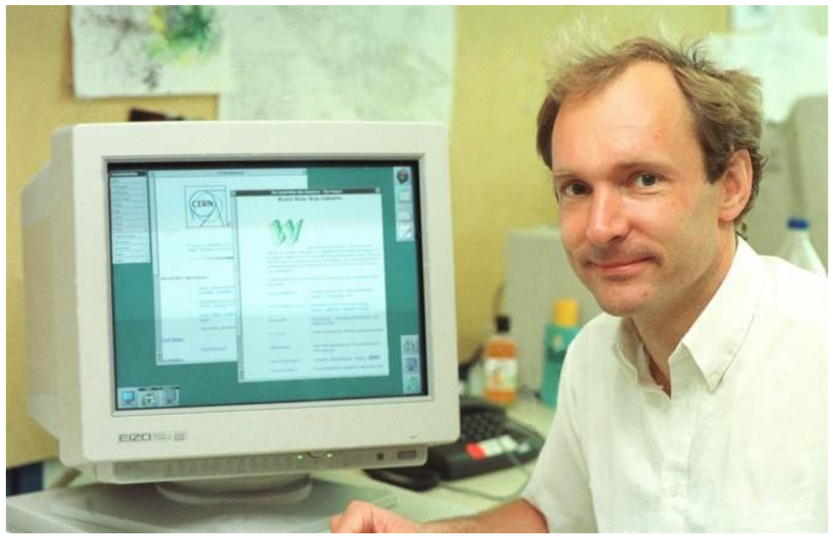
- try-catch 구문으로 async/await 형태 비동기 코드 에러 처리가 가능하다.
- async 함수는 catch 문을 사용해서 에러 처리를 하지 않으면 발생한 에러를 reject하는 프로미스를 반환한다.

02

HTTP 통신

HTTP 통신

- HTTP(Hyper Text Transfer Protocol)



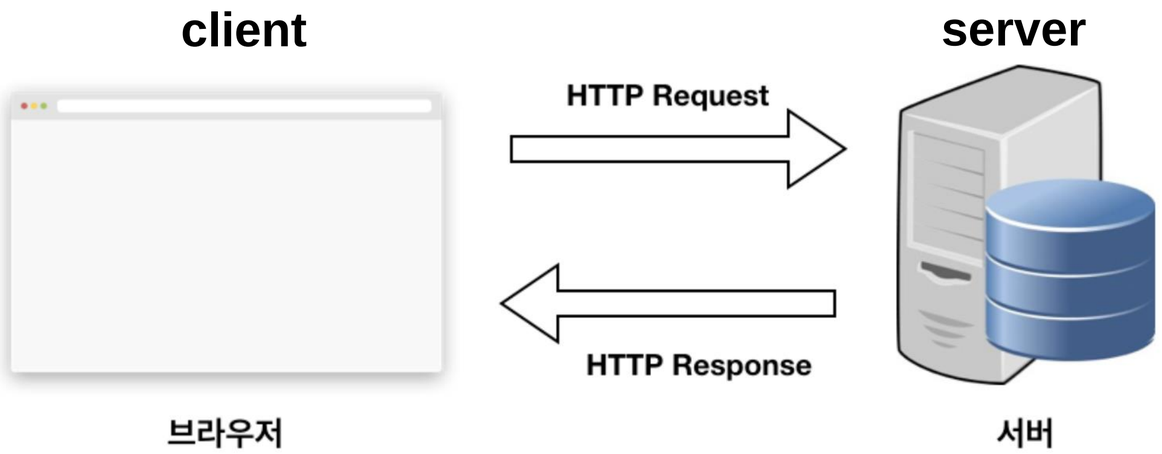
나 기억나?

HTTP 통신

- HTTP(Hyper Text Transfer Protocol)
  - 서버와 클라이언트 간의 메시지 교환 프로토콜

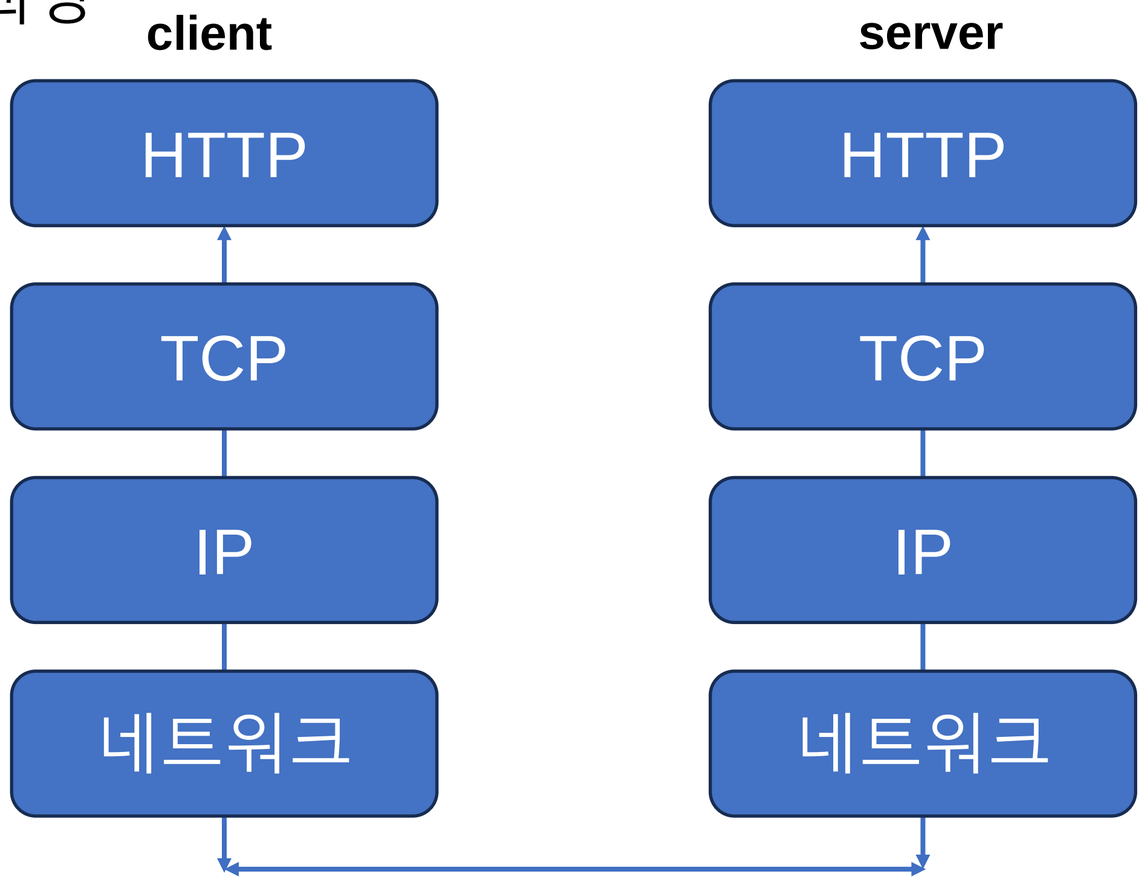
HTTP 통신

- Request / Response



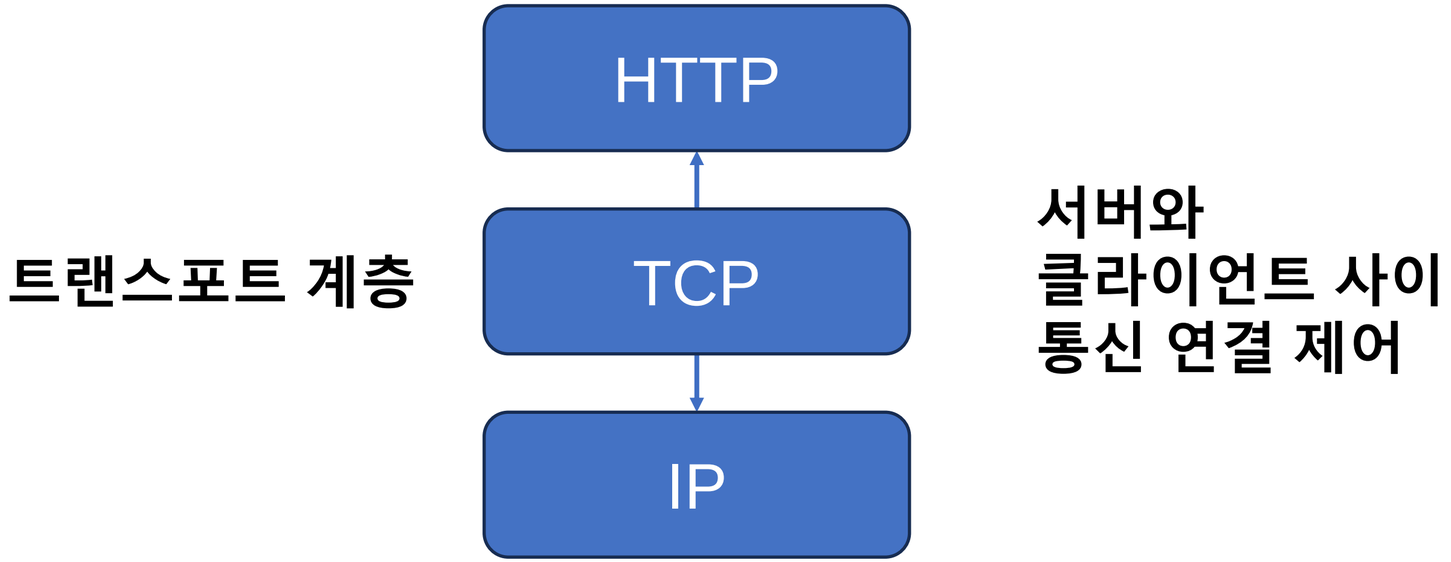
HTTP 통신

- 기기 간 통신 과정



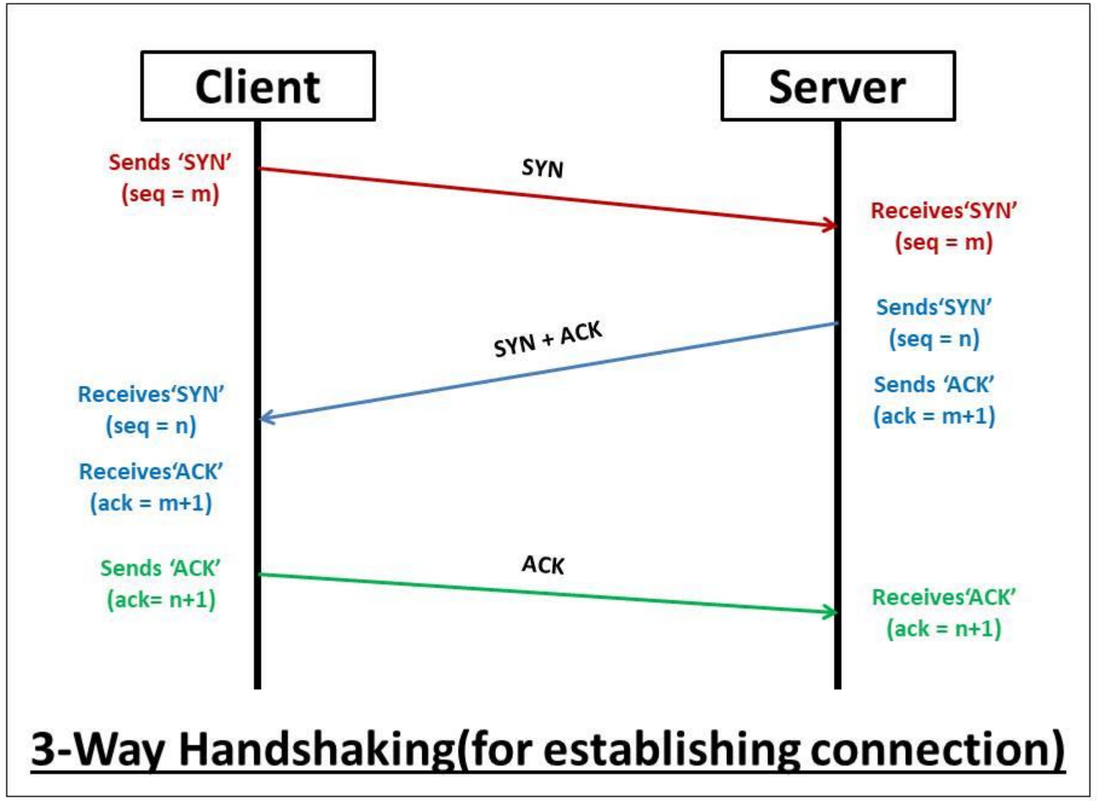
HTTP 통신

- TCP (Transmission Control Protocol)



HTTP 통신

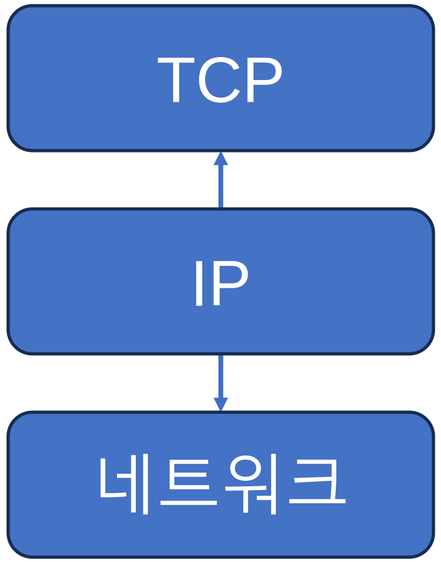
- TCP (Transmission Control Protocol)



HTTP 통신

- IP (Internet Protocol)

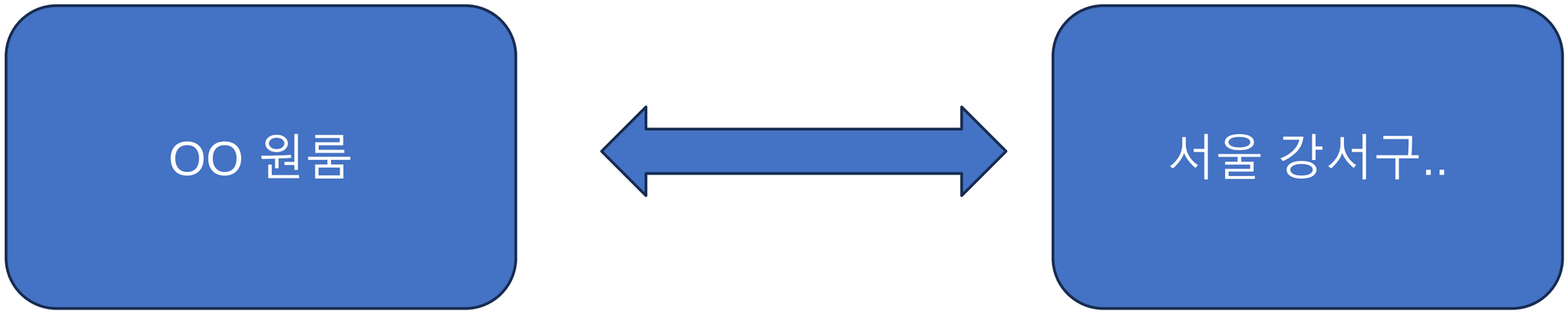
인터넷 계층



데이터 패킷 전달

HTTP 통신

- DNS (Domain Name System)
  - 도메인 이름 및 IP 주소를 확인하는 기능을 제공

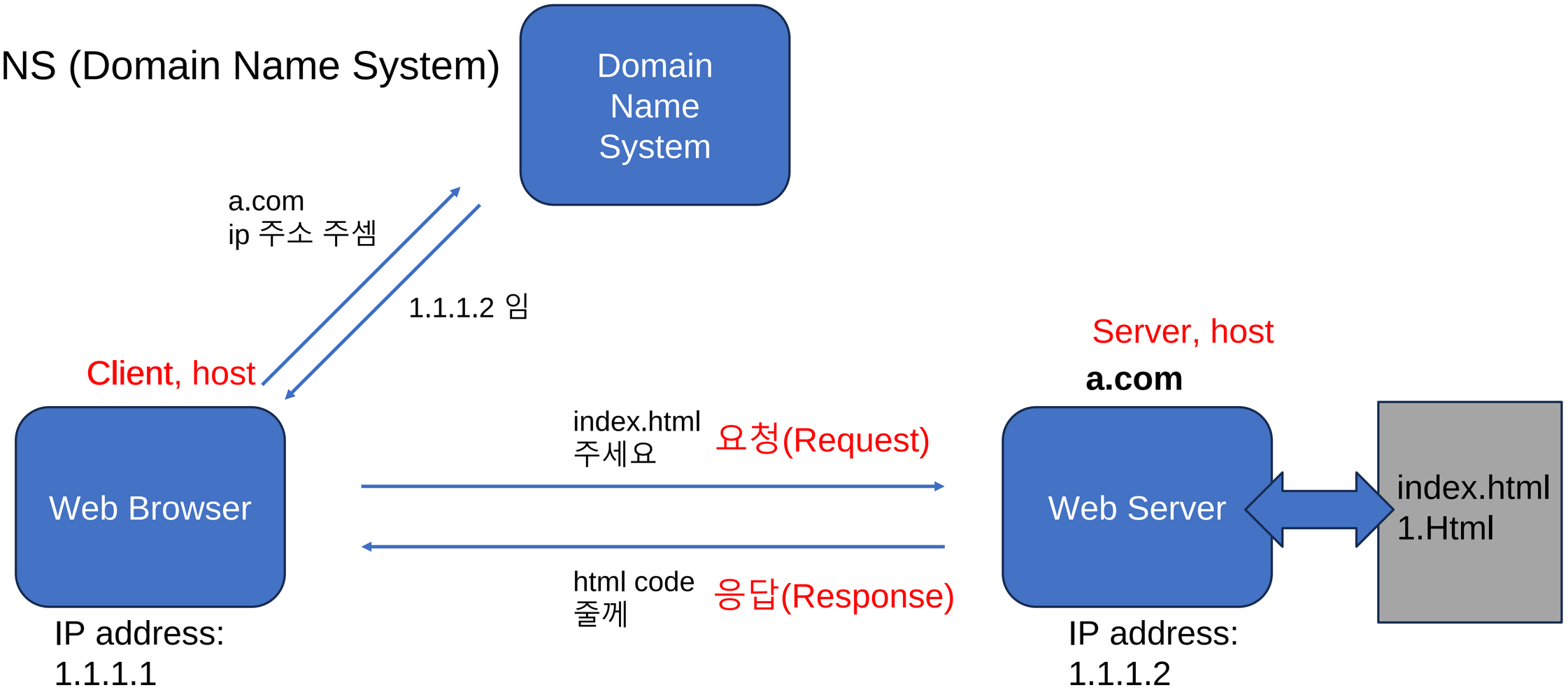


도메인
www.myWebSite.com

IP 주소
127.0.0.111

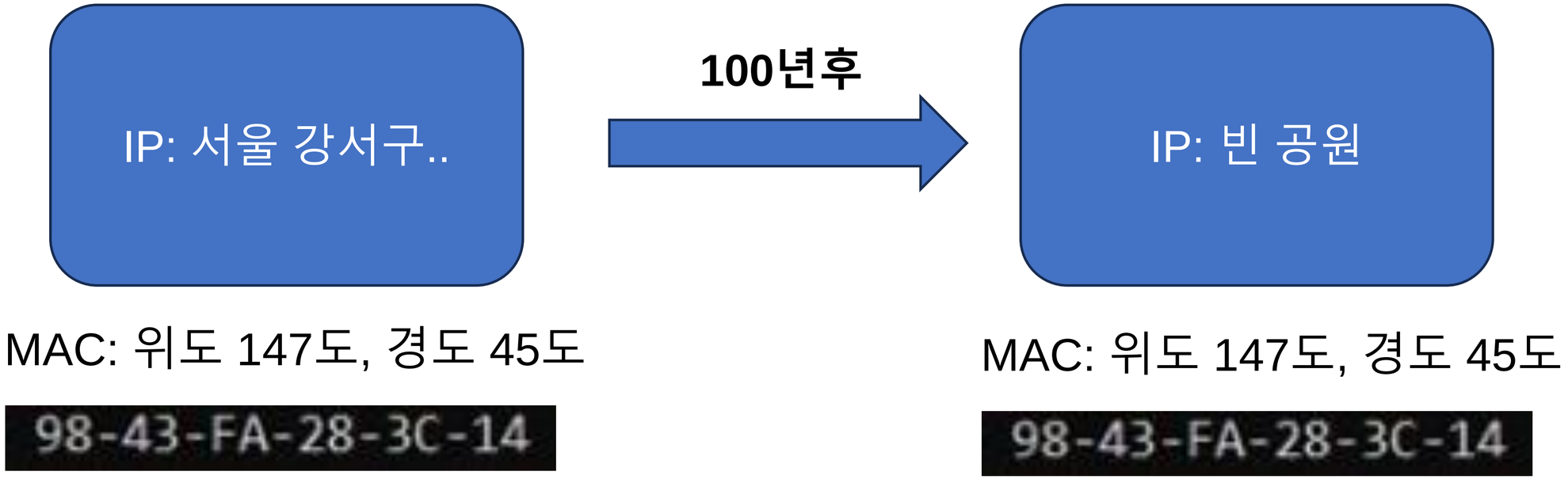
HTTP 통신

- DNS (Domain Name System)



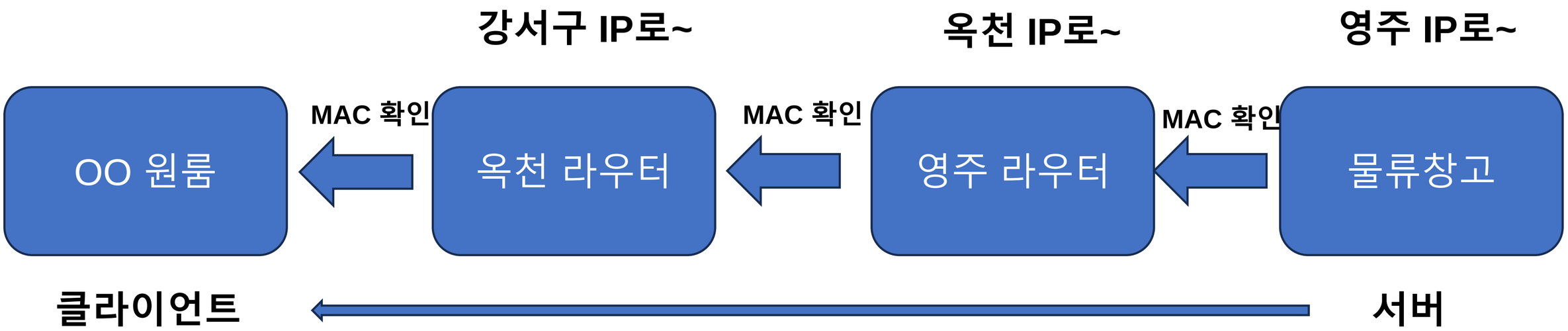
HTTP 통신

- IP 주소 / MAC 주소



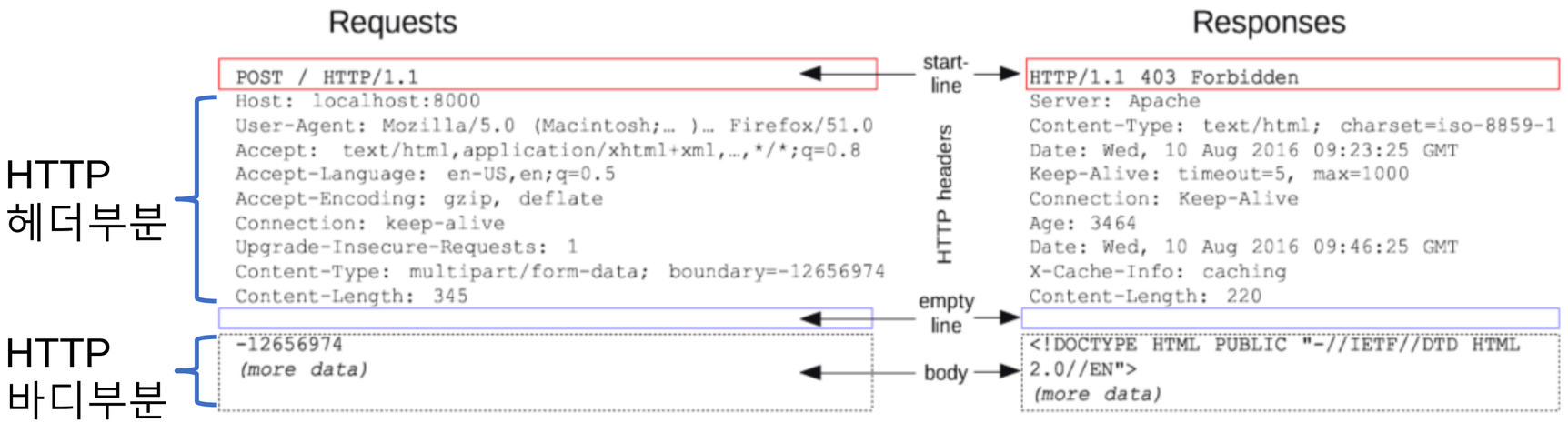
HTTP 통신

- ARP (Address Resolution Protocol): 주소를 찾아가는 프로토콜



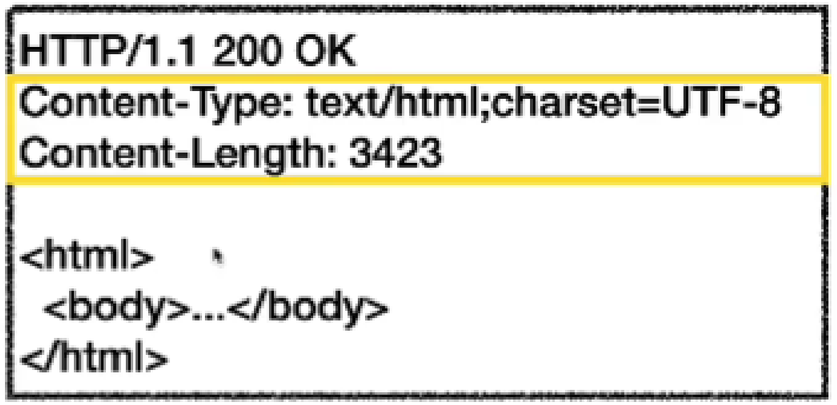
HTTP 통신

- HTTP 메시지
  - URI, 메서드, 프로토콜 버전, 상태 코드, 헤더, 바디 등이 포함



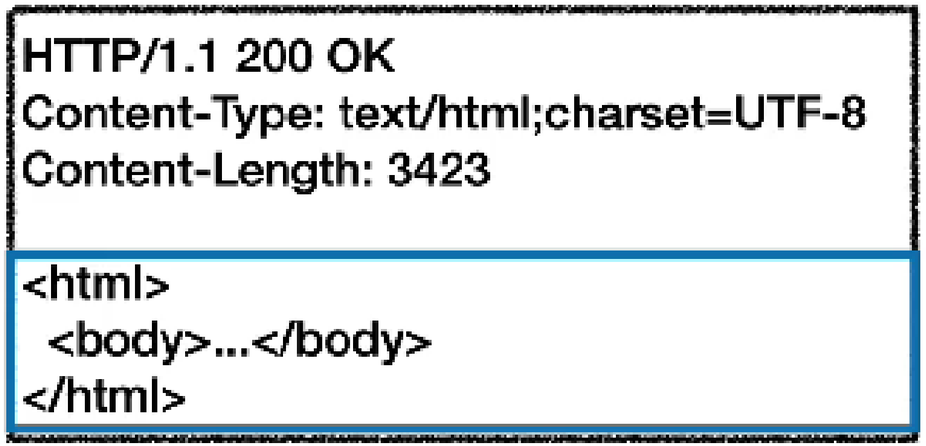
HTTP 통신

- HTTP 헤더
  - HTTP 전송에 필요한 모든 부가정보
  - 콘텐츠 관련 정보, 인증 관련 정보, 쿠키 정보, 캐시 관련 정보 등 서버와 클라이언트 간 통신 시 필요한 정보를 담는다.
    - 예) 메시지 바디의 내용, 메시지 바디의 크기, 압축, 인증 요청 클라이언트(브라우저) 정보, 서버 애플리케이션 정보, 캐시 관리 정보…
  - 표준 헤더가 너무 많음
    - https://en.wikipedia.org/wiki/List_of_HTTP_header_fields
  - 사용자 정의 헤더를 필요에 따라 추가할 수 있음



HTTP 통신

- HTTP 바디
  - 실제 전송할 데이터가 들어있음
  - HTML 문서, 이미지, 영상, JSON 등등 byte로 표현할 수 있는 모든 데이터 전송 가능



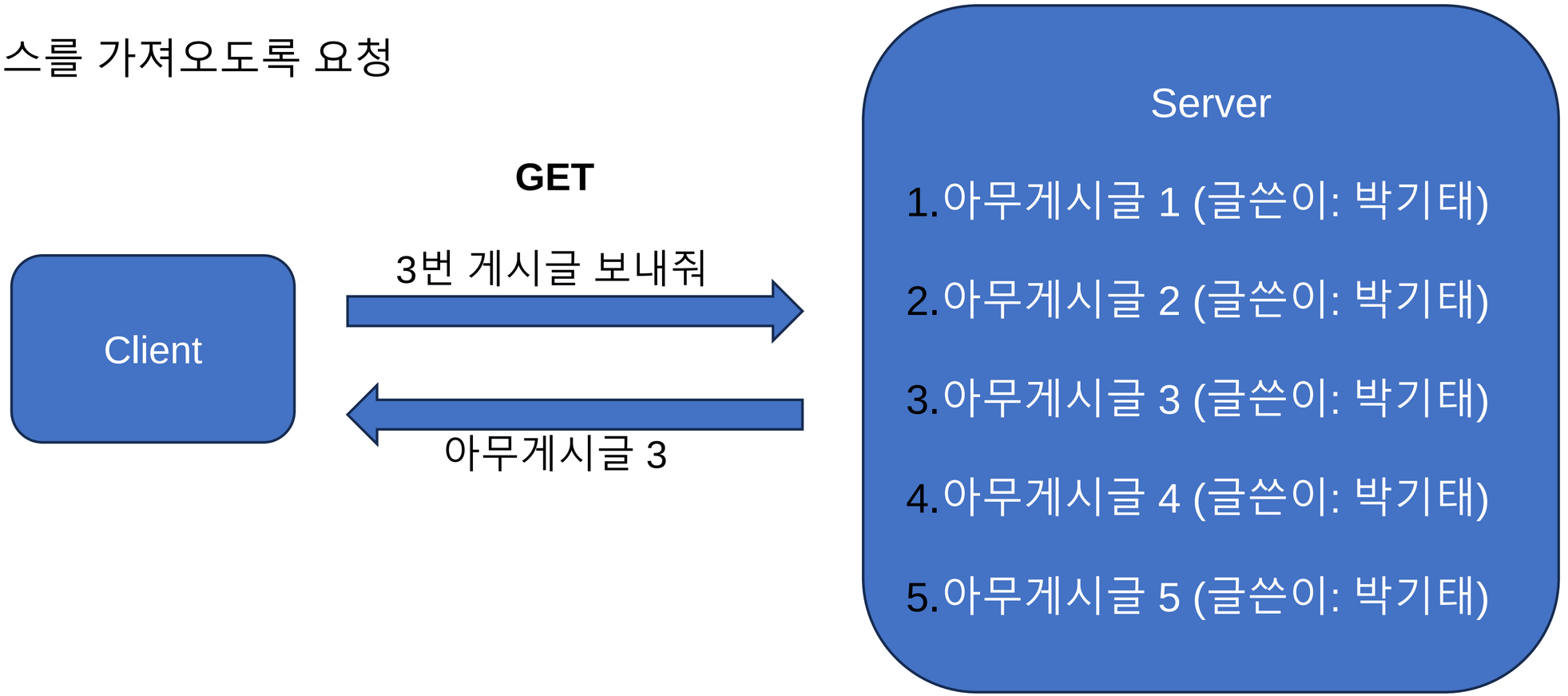
HTTP 통신

- HTTP 요청 메서드 종류

HTTP 요청 메서드	목적
GET	리소스 조회
POST	요청 데이터 처리, 주로 등록에 사용
PUT	리소스를 대체, 해당 리소스가 없으면 생성
PATCH	리소스 부분 변경
DELETE	리소스 삭제

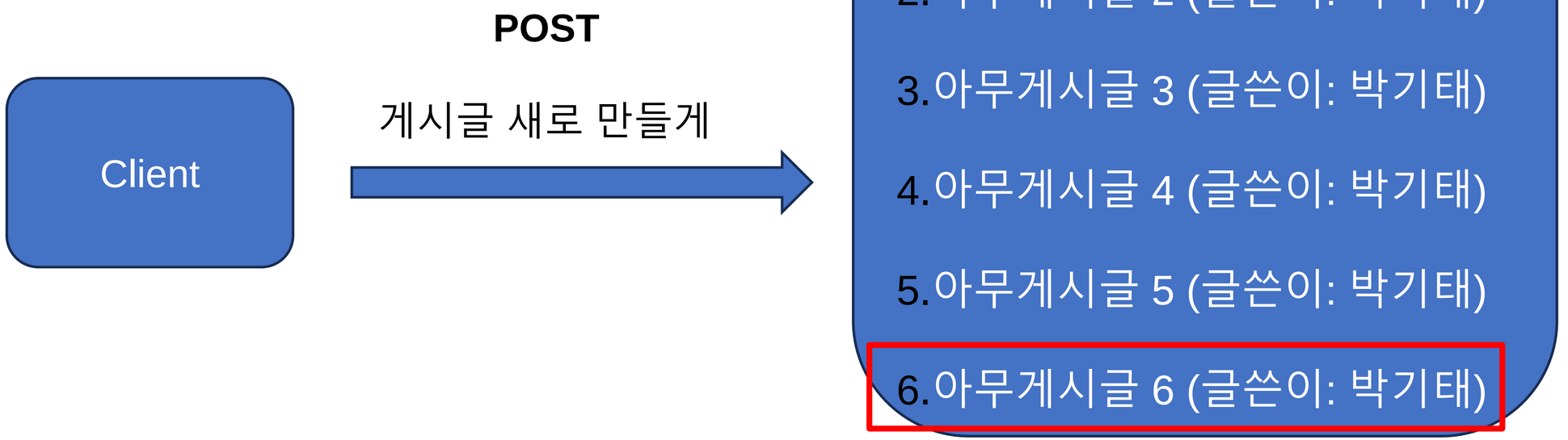
HTTP 통신

- GET 메서드
  - 특정 리소스를 가져오도록 요청



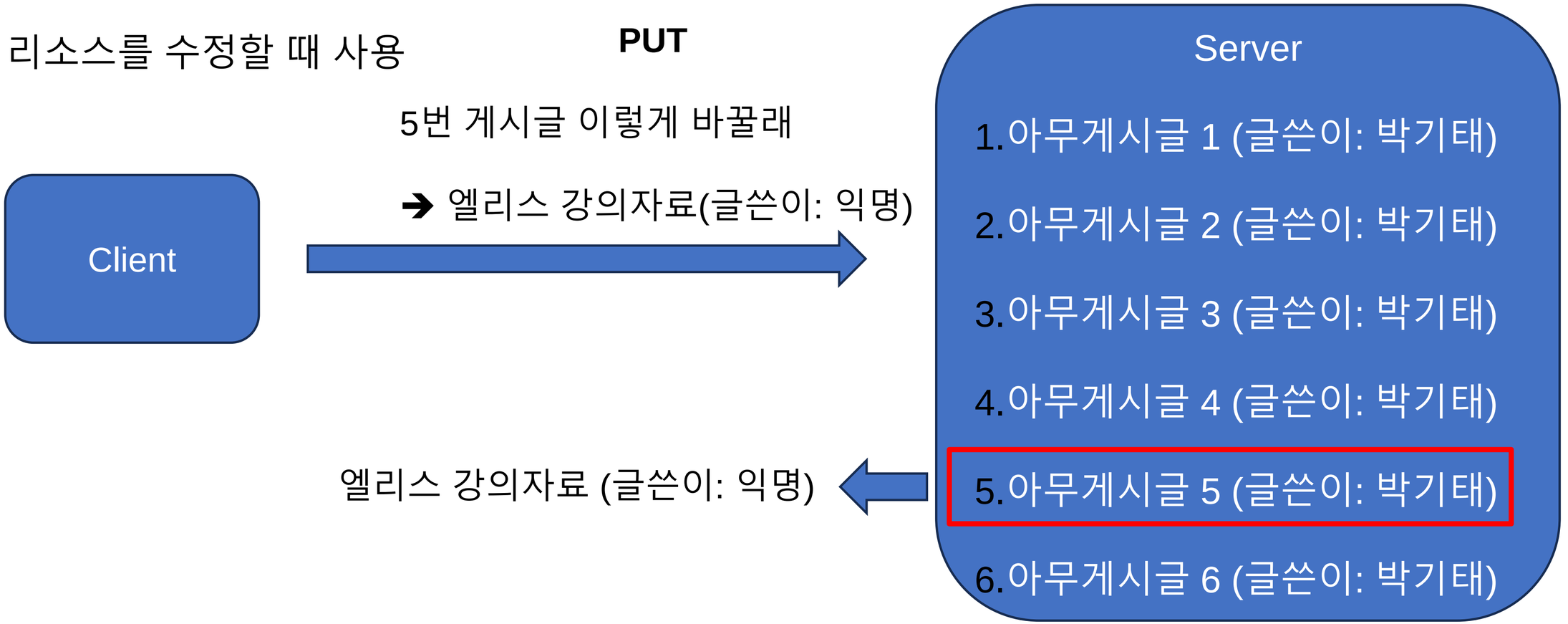
HTTP 통신

- POST 메서드
  - 대상 리소스의 요청 본문 내용을
    리소스의 시맨틱에 따라 처리하도록 요청
    - 우리가 보낼 리소스의 본문 내용을
      우리가 처리하고 싶은 방식으로 처리해주세요



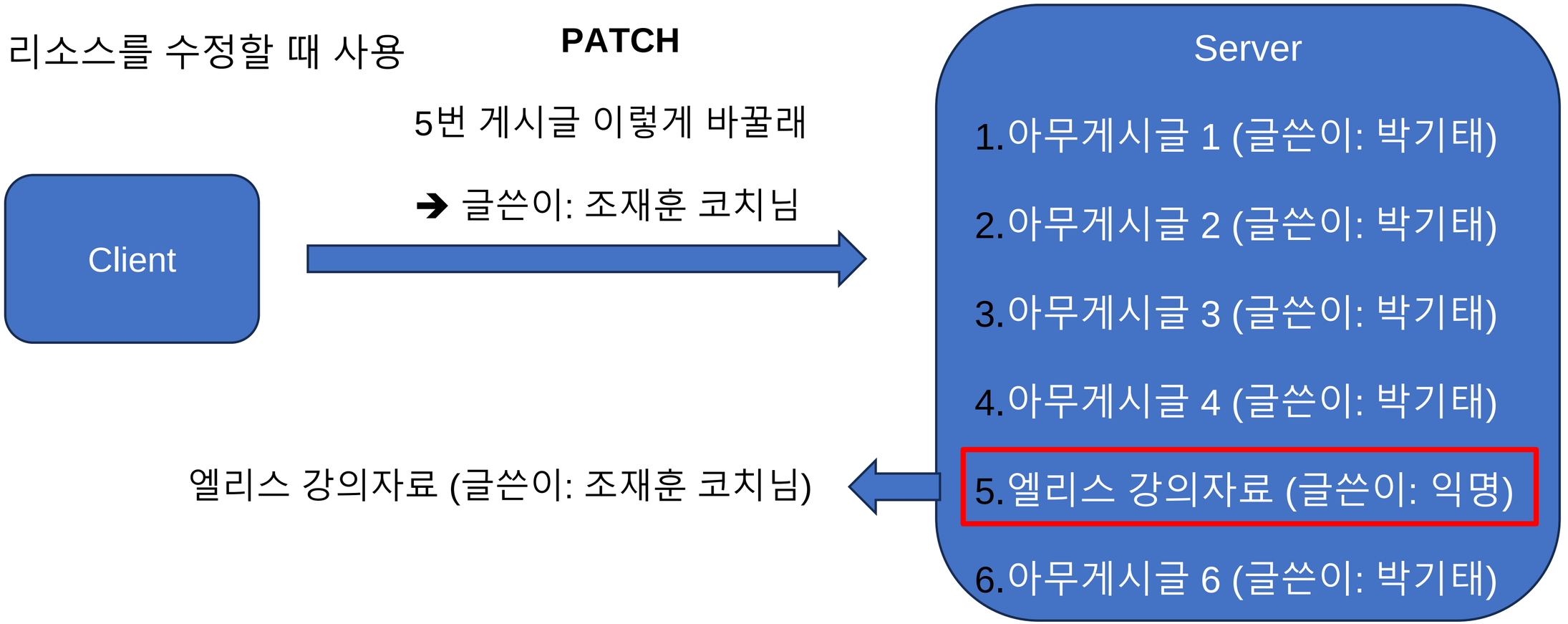
HTTP 통신

- PUT / PATCH
  - 리소스를 수정할 때 사용



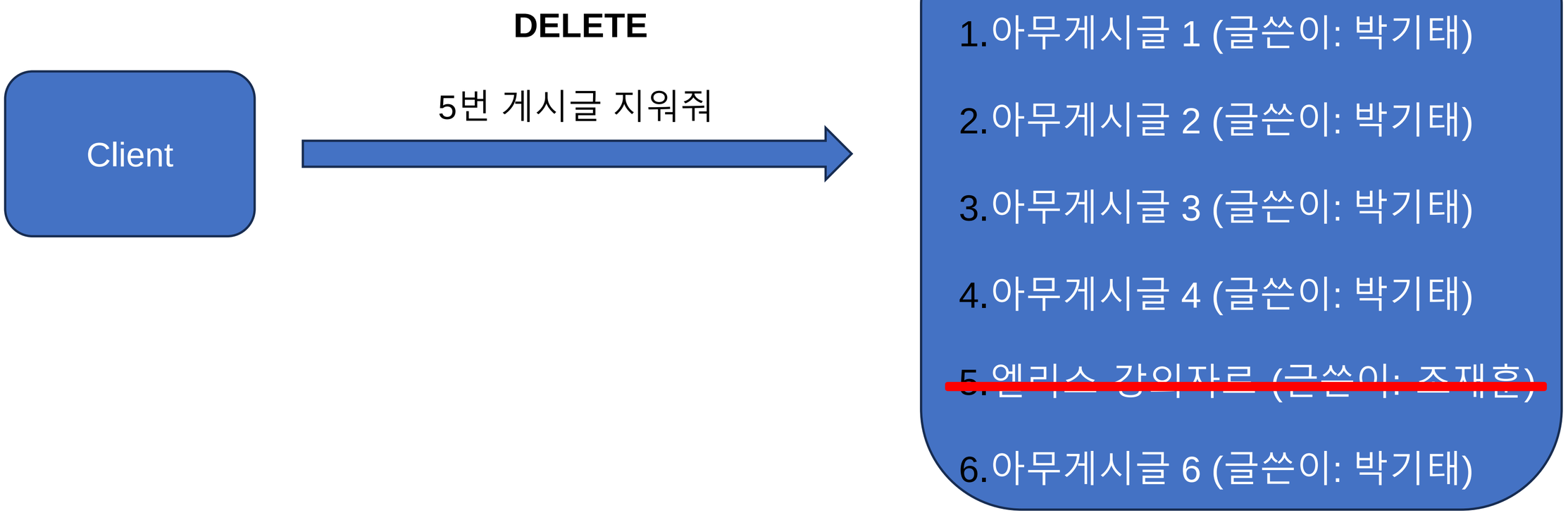
HTTP 통신

- PUT / PATCH
  - 리소스를 수정할 때 사용



HTTP 통신

- DELETE
  - 지정한 리소스를 삭제

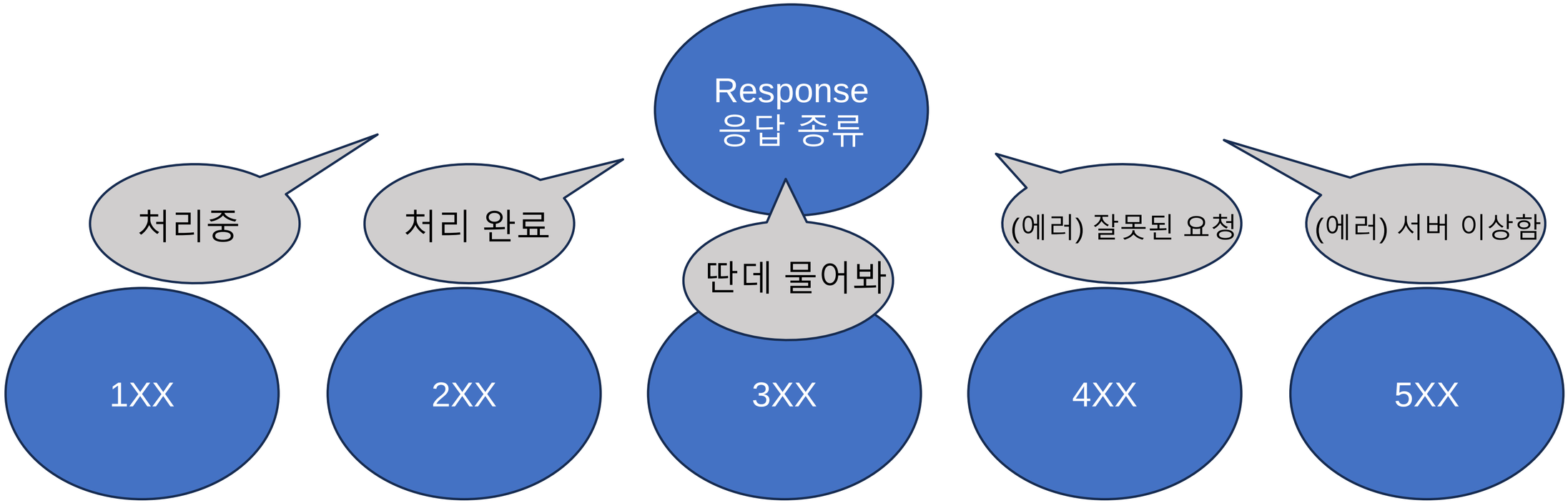


HTTP 통신

- 그 외 메서드
  - CONNECT, HEAD, OPTIONS, TRACE

HTTP 통신

- HTTP 상태코드
  - HTTP 요청 시, 클라이언트는 요청의 결과에 대한 상태 정보를 서버로부터 얻는다



HTTP 통신

- HTTP 응답 종류
  - HTTP 요청 시, 클라이언트는 요청의 결과에 대한 상태 정보를 서버로부터 얻는다
    - 2XX: 성공
      - 200 (OK): 서버가 요청을 제대로 처리했다는 뜻
      - 201 (created): 요청을 처리하였고 요청에 따른 새로운 리소스가 서버에 저장됨
      - 204 (No content): 서버가 요청을 제대로 처리했는데 요청에 따른 콘텐츠를 제공하지 않을 때 사용
    - 3XX: 추가적인 동작이 필요

HTTP 통신

- HTTP 응답 종류 (3XX)

상태 코드	이름	Redirect	POST -> GET	URL 영구 이동
301	moved Permanently	O	X	O
302	Found	O	X	X
303	See Other	O	O	X
304	Not Modified	X	X	X
307	Temporary Redirect	O	X, 메소드 변경 금지	X
308	Permanent Redirect	O	X, 메소드 변경 금지	O

HTTP 통신

- HTTP 응답 종류
  - HTTP 요청 시, 클라이언트는 요청의 결과에 대한 상태 정보를 서버로부터 얻는다
    - 4XX: 클라이언트 요청 오류
      - 400(Bad request): 서버가 요청 구문을 인식하지 못했다는 뜻
      - 401(Unauthorized): 해당 요청을 들어주려면 인증이 필요하다는 뜻 (예: 로그인 실패)
      - 403(Forbidden): 클라이언트가 리소스에 대한 필요 권한을 갖고 있지 않다는 뜻
      - 404(Not found): 해당 페이지를 찾을 수 없거나 요청을 거부했는데 그 사유에 대해 비밀로 처리할 때 쓰임
    - 5XX: 서버 내부 오류
      - 500(Internal Server Error): 서버에 오류가 발생하여 요청을 수행할 수 없을 때 사용
      - 501(Not Implemented): 서버에 요청을 수행할 수 있는 기능이 없을 때 사용
      - 503(Service Unavailabe): 서버가 다운되어 사용할 수 없을 때 사용 (예: 서버의 유지보수로 작동이 중단되거나 과부하가 걸렸을 때)

03

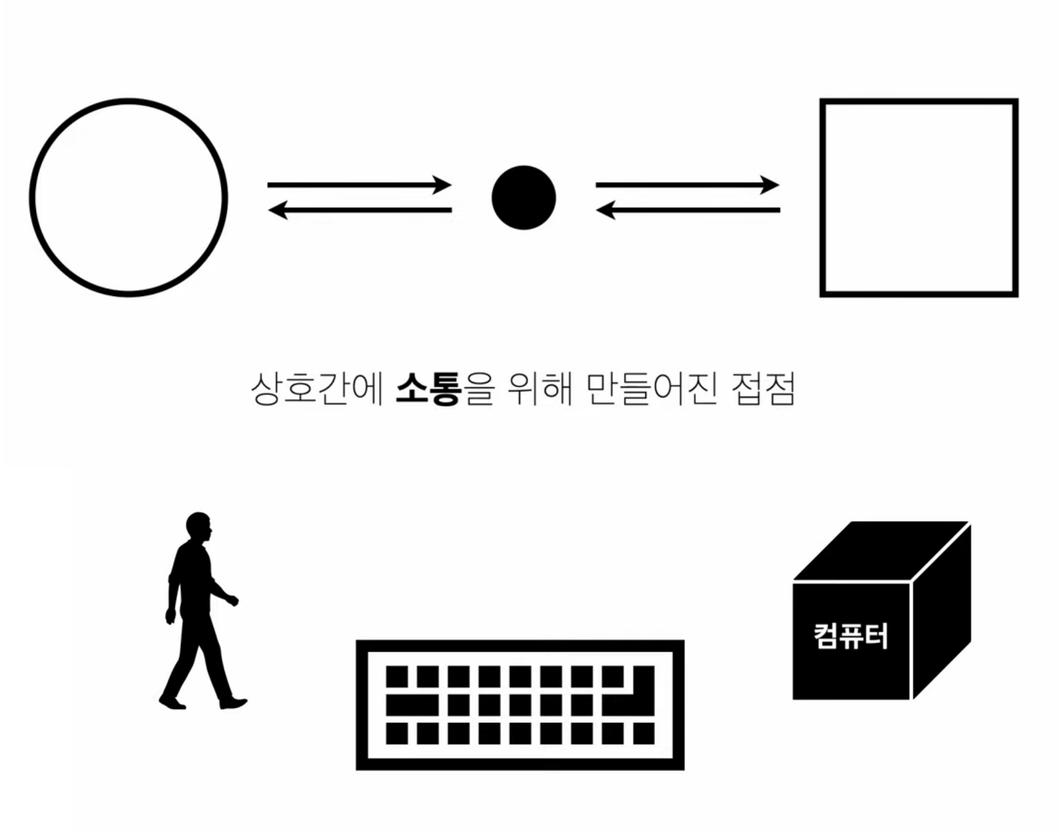
fetch API

API

- API (Application Programming Interface)
  - 응용 프로그램에서 사용할 수 있도록
    운영체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 만든 인터페이스

API

- 인터페이스



API

- 유저 인터페이스(User Interface)

API

- API (Application Programming Interface)

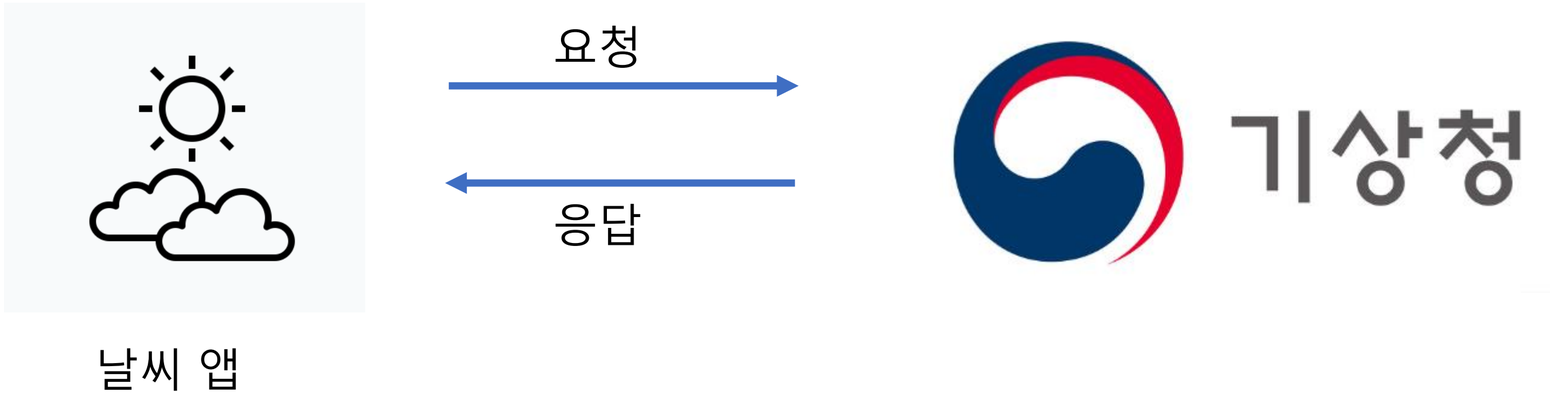
Application Programming Interface

응용 프로그램에서 소통하기 위한 접점

➔ 응용 프로그램에서 데이터를 읽거나 쓰기 위해 사용하는 인터페이스

API

- API: 응용 프로그램, 애플리케이션에서 데이터를 읽거나 쓰기 위해 사용하는 인터페이스



API

- API: 응용 프로그램, 애플리케이션에서 데이터를 읽거나 쓰기 위해 사용하는 인터페이스

날씨 앱

요청

응답

```
{
  “today”: “2023-06-05”
  “weather”: “sunny”
}
```

기상청 API:
http://api.data.go.kr/weather/list

기상청

Fetch API

- HTTP 요청 전송 기능을 제공하는 Web API
- 프로미스를 지원한다.
  - 네트워크 요청 성공 시, response 객체 resolve
  - 네트워크 요청 실패 시, 에러 reject
- 문법
  - fetch(url, options)
    - url은 웹 url 이 아닌 로컬 디렉토리 경로도 된다.

```
let result = fetch(serverURL)

result
  .then(response => {
    if (response.ok) {
      // 요청 성공.
    }
  })
  .catch(error => {
    // 요청 실패.
  })
```

Fetch API

- Response 객체는 결과에 다양한 결과를 담음

```
fetch(serverURL)
  .then(response => {
    response.ok
    response.status
    response.statusText
    response.url
    response.bodyUsed
  })
```

Fetch API

- Response 객체는 결과에 다양한 결과를 담음
  - response.json() 메서드는 얻어온 body 정보를 json으로 만드는 Promise를 반환한다.

```
fetch(serverURL)
  .then(response => {
    return response.json()
  })
  .then(json => {
    console.log('body : ',
json)
  })
```

Fetch API

- fetch 메서드 옵션
  - method 필드로 여러 요청 메서드를 활용한다.
  - headers, body 필드를 활용해 서버에 추가 정보를 보낸다

```
fetch(serverURL, {
  method: 'post',
  headers: {
    'Content-Type':
'application/json;charset=utf-8',
    Authentication: 'mysecret'
  },
  body: JSON.stringify(formData)
})
  .then(response => {
    return response.json()
  })
  .then(json => {
    console.log('POST 요청 결과:', json)
  })
```

참고

- 모던 자바스크립트 Deep Dive
- 엘리스 이론 자료
- 모든 개발자를 위한 HTTP 웹 기본 지식 / 김영한
- 우아한테크
- https://ko.javascript.info/async-await
- https://learnjs.vlpt.us/async/02-async-await.html
- https://untitledtblog.tistory.com/98
- https://dongchans.github.io