

# #01 ML 프로그램 클리닉 I



Web System

01

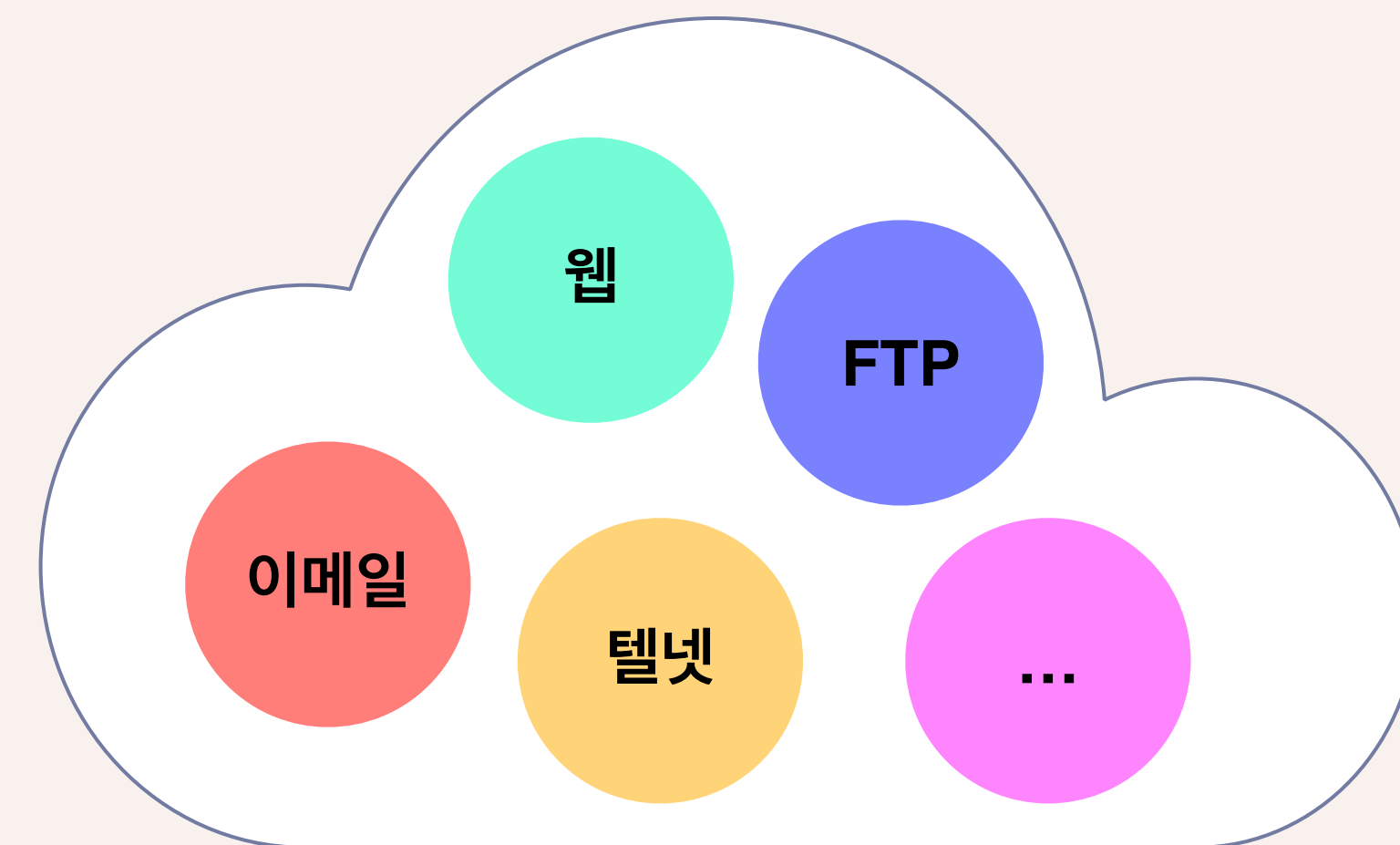
# 인터넷과 웹

## ✓ 인터넷

- 컴퓨터가 서로 연결되어 TCP/IP 통신 프로토콜을 이용하여 정보를 주고 받는 컴퓨터 네트워크

## ✓ 웹

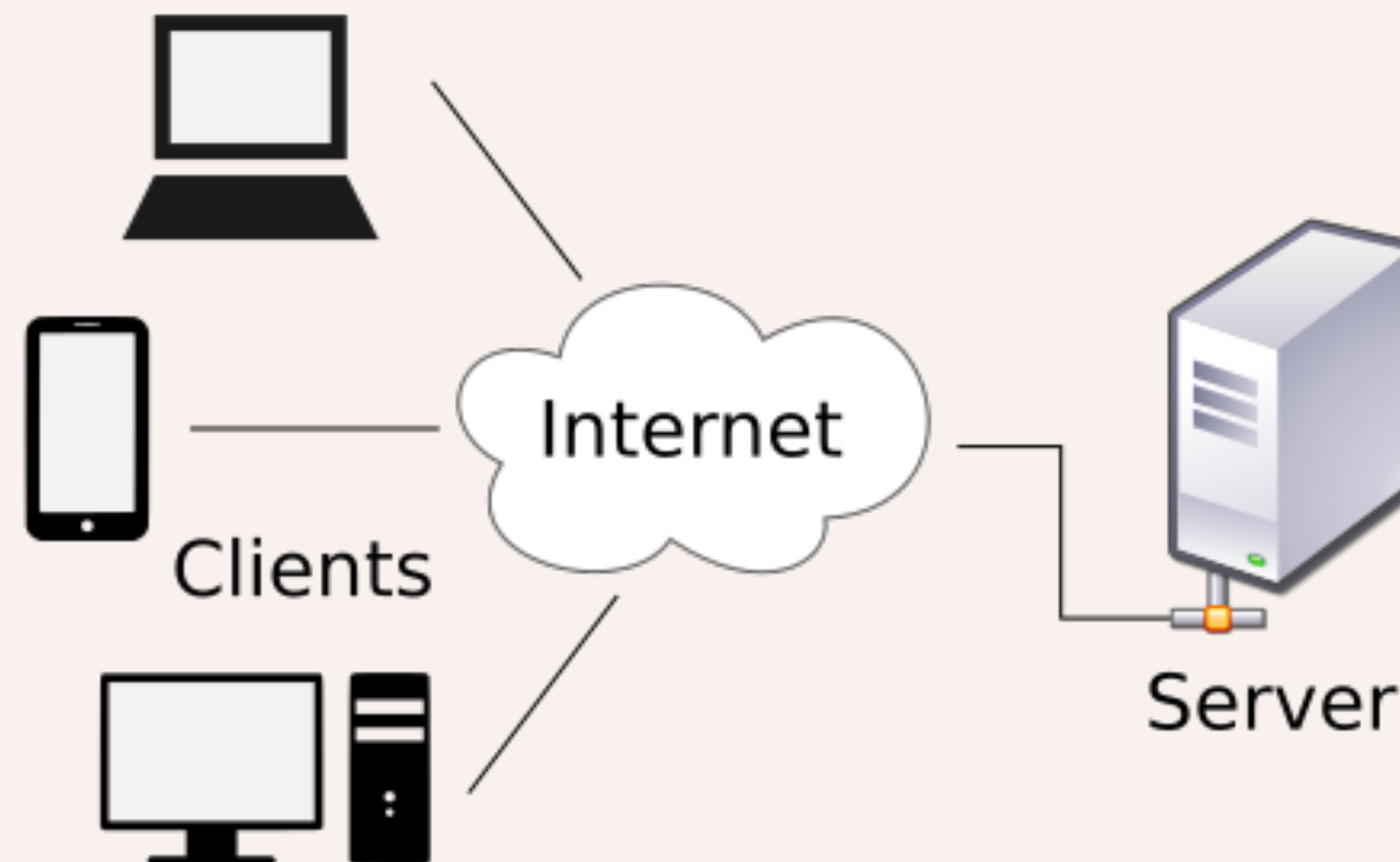
- 인터넷에 연결된 컴퓨터들을 통해 사람들이 정보를 공유할 수 있는 정보 공간으로 월드 와이드 웹 (world wide web)의 줄임말



# 웹의 동작 원리

## ✓ Client-Server 시스템

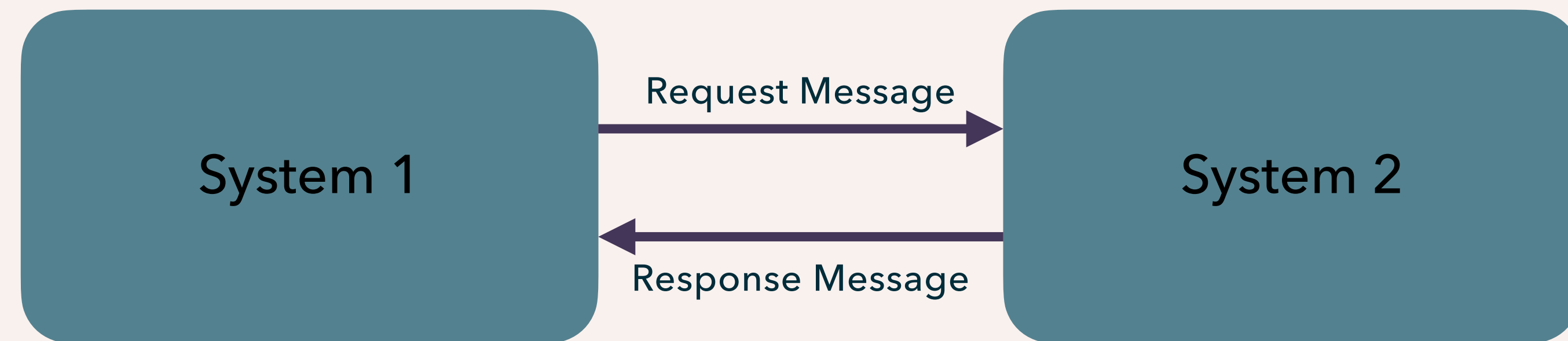
- 클라이언트와 서버로 나누는 네트워크 아키텍처를 의미
- 사용자 PC에는 클라이언트 응용프로그램을 설치하여 화면을 처리하고, 서버에는 자료를 처리하는 응용 프로그램을 설치하여 클라이언트와 서버 간 통신 프로토콜을 이용하여 동작하는 시스템
- 서버라고 하는 자원 또는 서비스 공급자와 클라이언트라고 하는 서비스 요청자 간에 작업 또는 작업 부하를 분할하는 분산 응용 시스템



# 웹의 동작 원리

## ✓ Request-Response Messaging Pattern

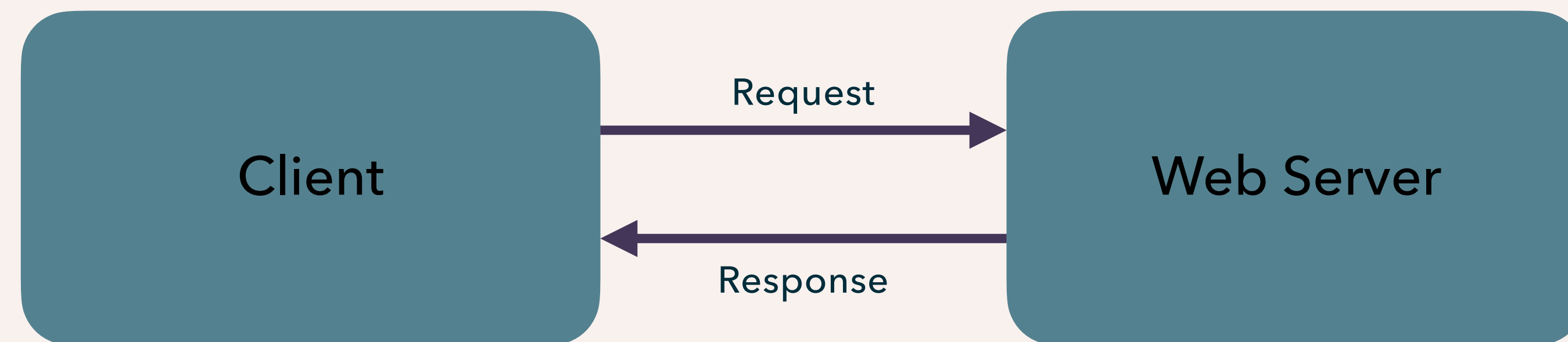
- 요청자가 응답자 시스템에 요청(request) 메시지를 전송
- 응답자는 그 메시지에 대한 처리를 한 응답(response) 메시지를 요청자에게 반환하는 메시지 교환 패턴



# 웹의 동작 원리

## ✓ 웹 시스템 구조

- Client-Server System의 대표적인 예
- 클라이언트가 특정 페이지를 웹 서버에 URL로 요청(request)
- 웹 서버가 요청을 처리한 후 그 결과를 클라이언트에게 HTML문서로 응답(response)



# 웹 서버의 구조

## ✓ 웹 서버 (Web Server)

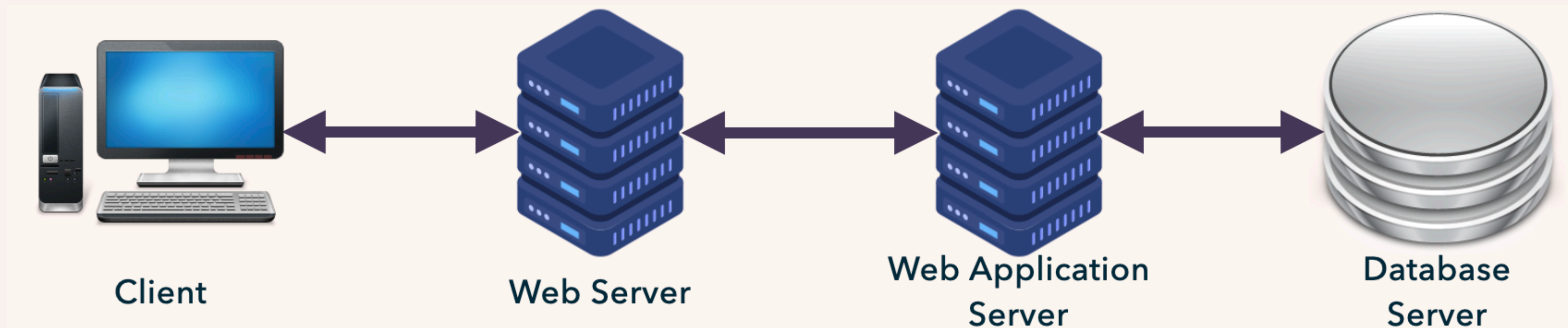
- 클라이언트로부터 직접 요청을 받아 처리하는 서버, 정적 웹 페이지들을 제공

## ✓ 웹 애플리케이션 서버 (WAS, Web Application Server)

- 웹 서버로부터 요청을 받아 데이터 가공 작업을 수행, 동적 웹 페이지들을 제공

## ✓ 데이터베이스 서버 (Database Server)

- 데이터베이스와 이를 관리하는 DBMS(Database Management Server)를 운영하는 서버

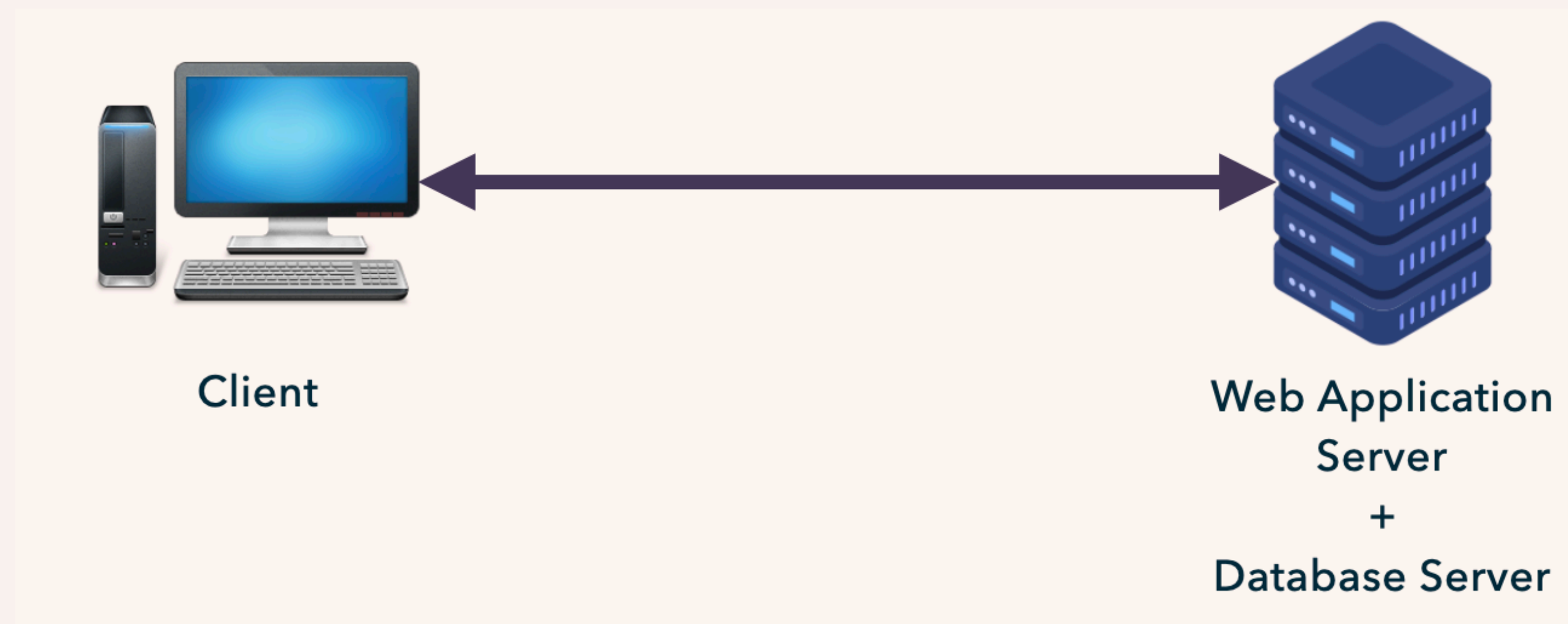




# 웹 서버의 물리적 분리에 따른 분류

## ✓ 1-Tier Architecture

- 웹 애플리케이션 서버와 데이터베이스 서버가 하나의 물리적 서버에서 동작하는 구조
- 웹 애플리케이션 서버가 웹 서버의 역할을 수행





# 웹 서버의 물리적 분리에 따른 분류

## ✓ 2-Tier Architecture

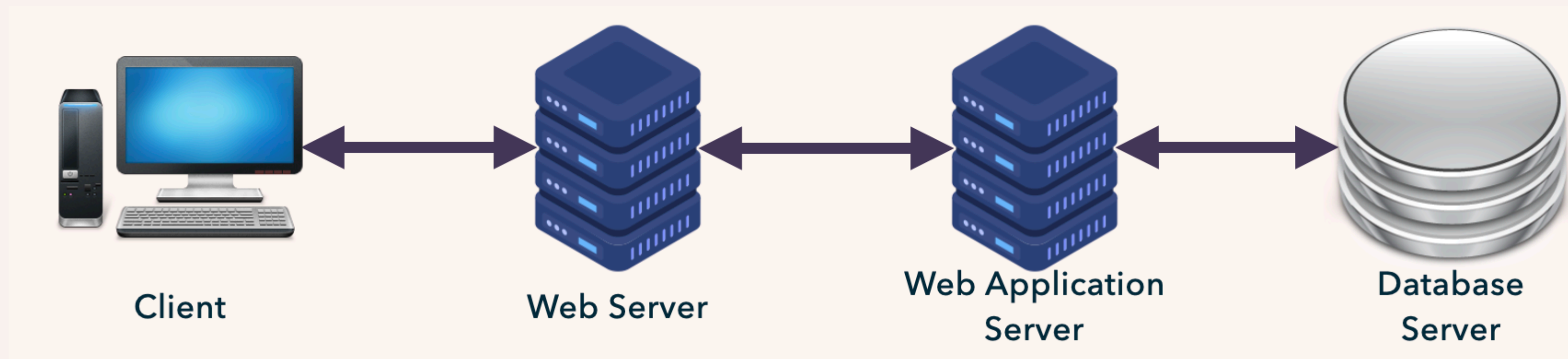
- 웹 애플리케이션 서버와 데이터베이스 서버가 각각 독립된 물리적 서버에서 동작하는 구조
- 웹 애플리케이션 서버가 웹 서버의 역할을 수행



# 웹 서버의 물리적 분리에 따른 분류

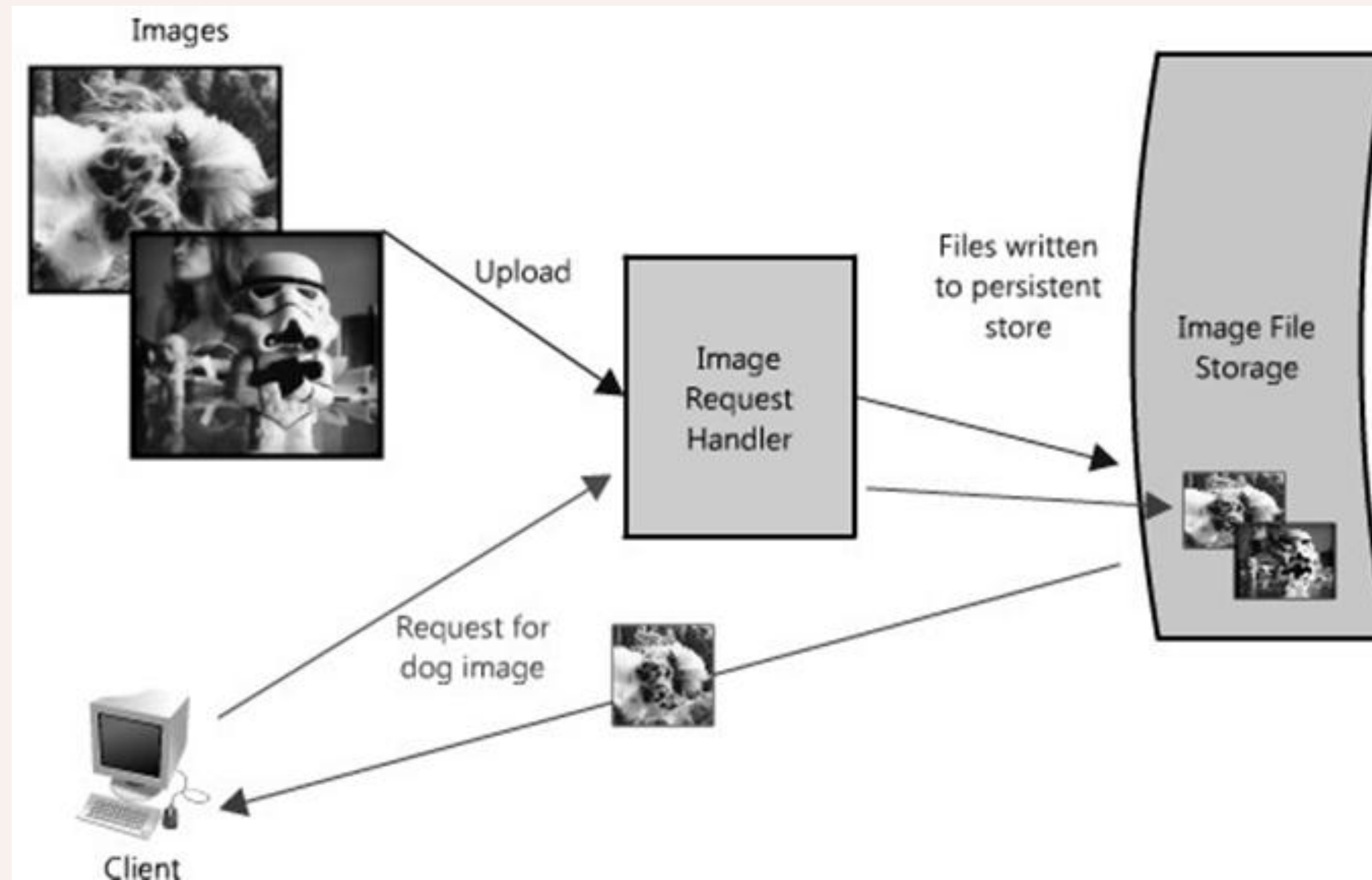
## ✓ N-Tier Architecture

- 웹 서버, 웹 애플리케이션 서버, 데이터베이스 서버가 각각 독립된 물리적 서버에서 동작되는 구조
- 일반적으로 비즈니스 로직을 담당하는 웹 애플리케이션 서버를 완전히 분리하여 데이터베이스 서버와 웹 서버 사이에 배치
- 일반적으로 3-Tier로 구성



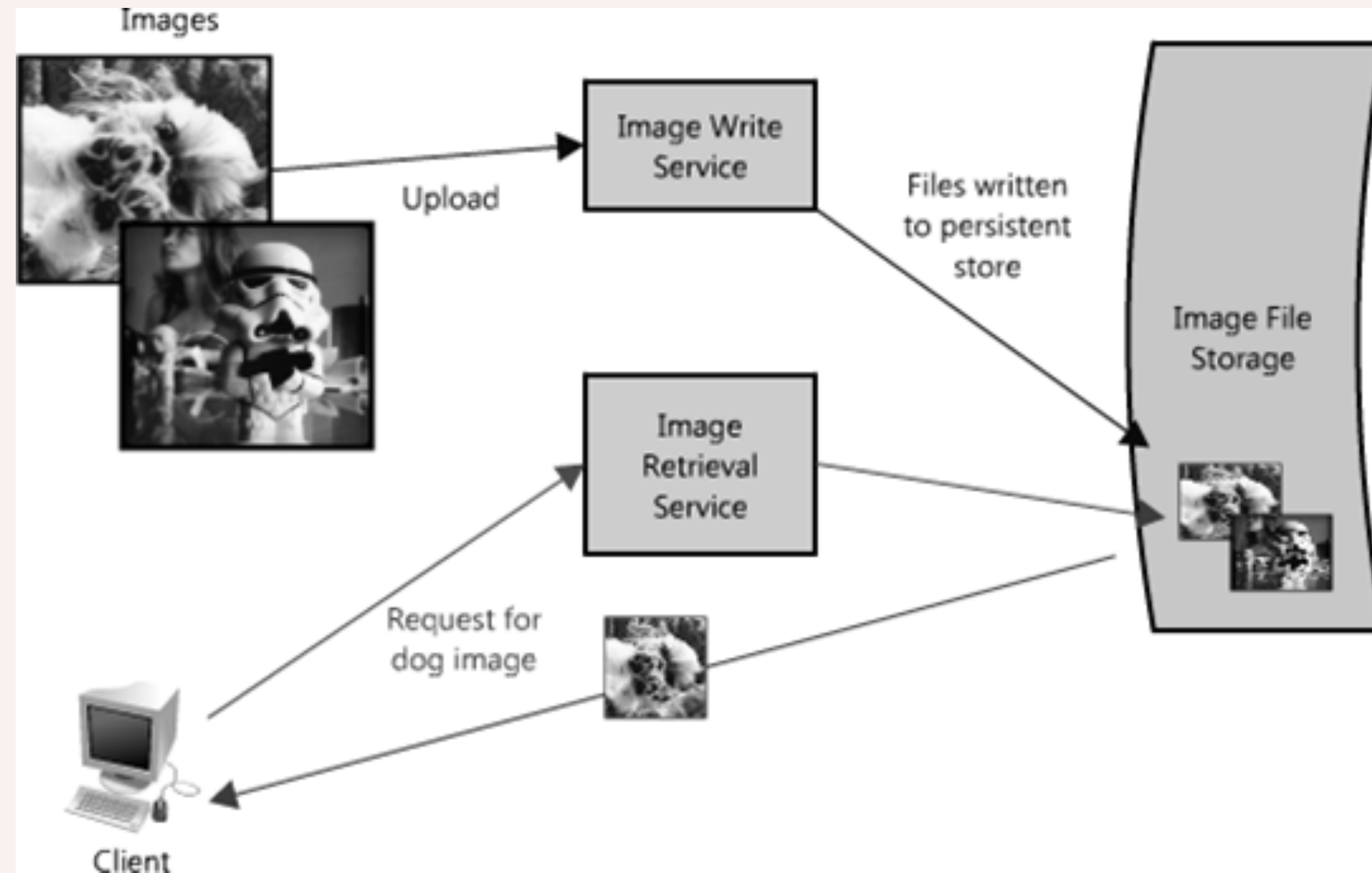
# 웹 서버의 다양한 분리 사례 1

✓ 이미지 호스팅 단일 시스템



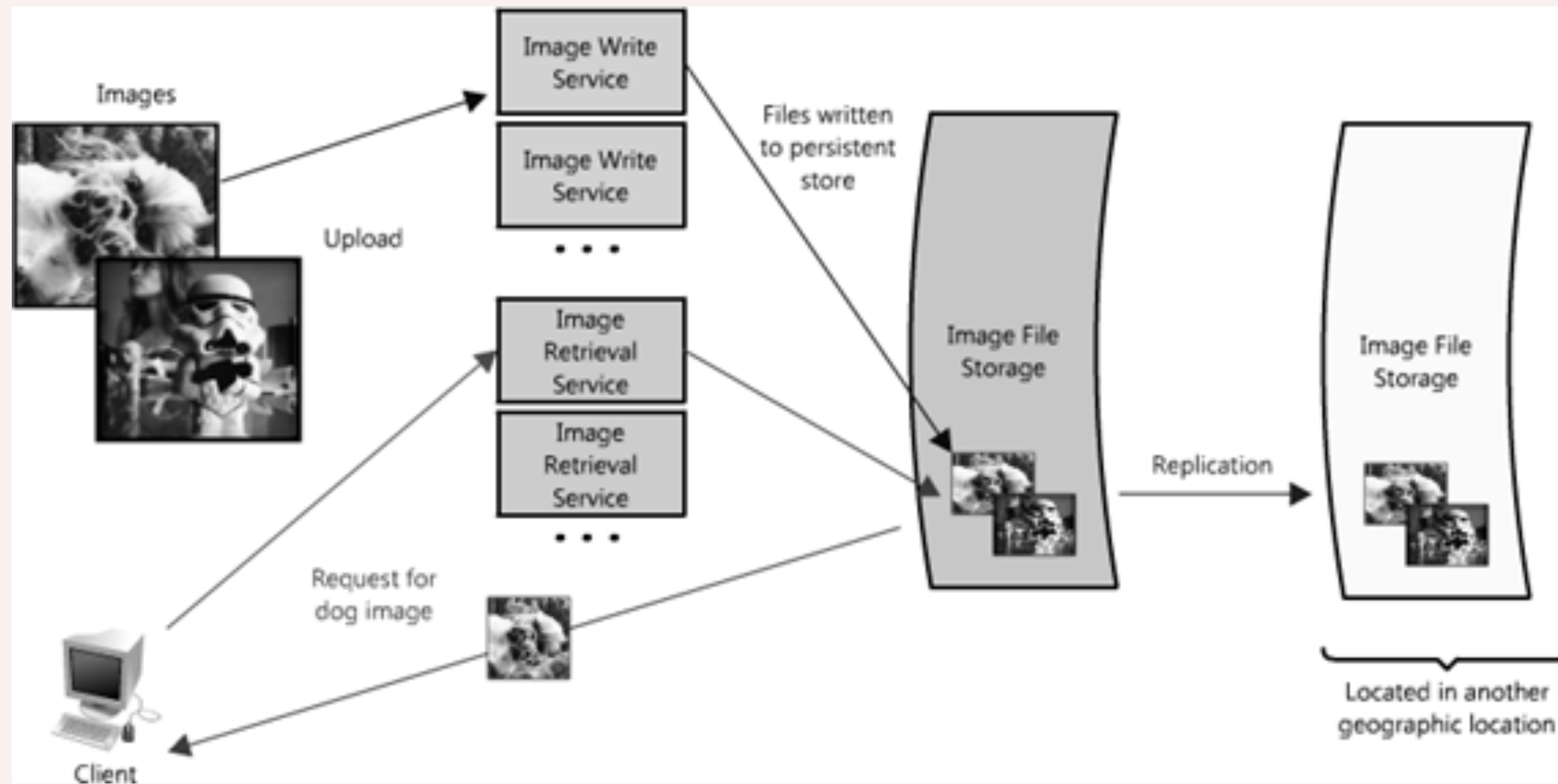
# 웹 서버의 다양한 분리 사례 2

✓ 이미지 호스팅 읽기와 쓰기 분리



# 웹 서버의 다양한 분리 사례 3

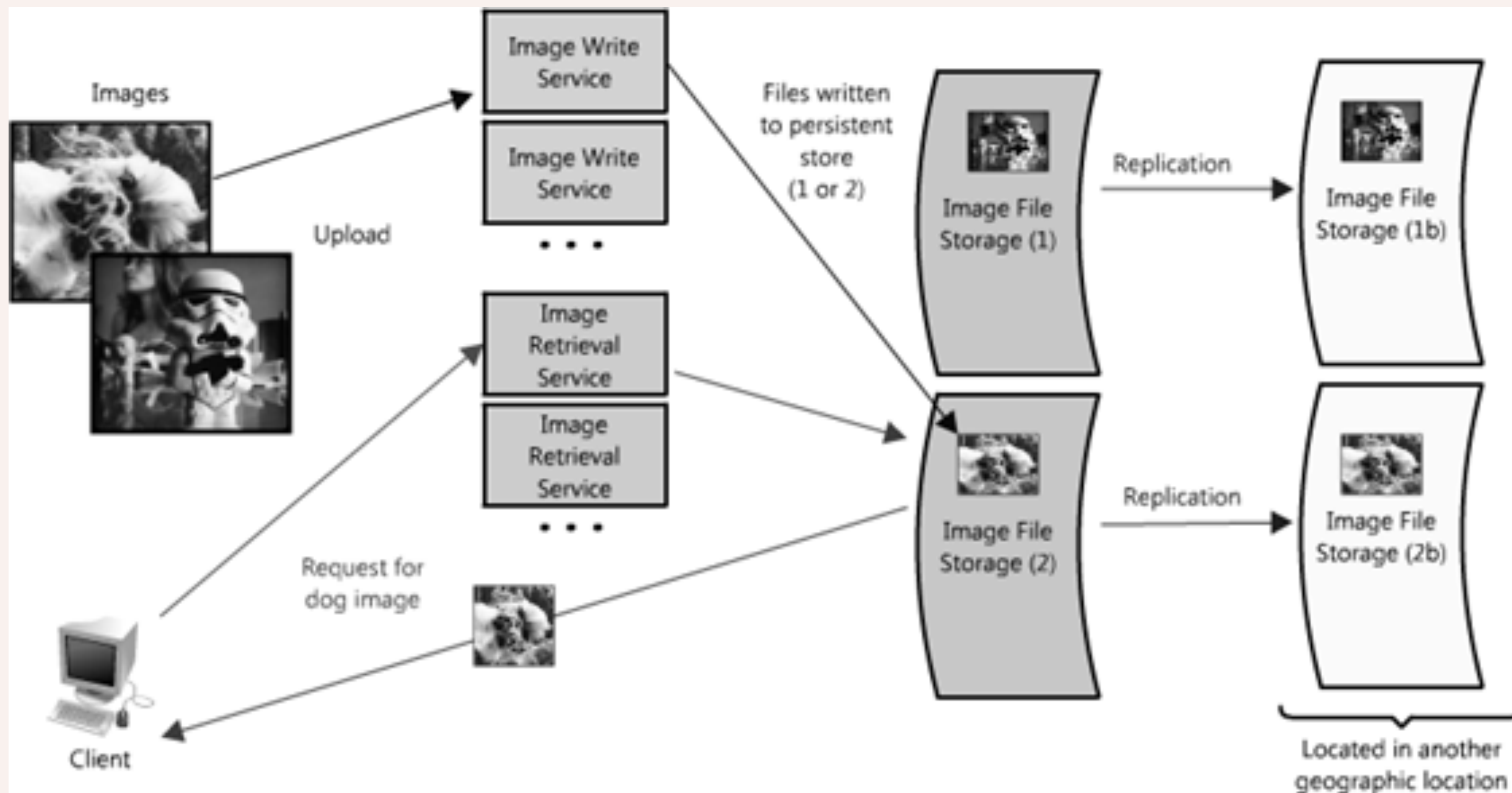
✓ 이미지 호스팅 이중화 고려





# 웹 서버의 다양한 분리 사례 4

✓ 이미지 호스팅 수평 확장(Scale out) 및 수직 확장(Scale up) 고려



# 정적 웹 페이지와 동적 웹 페이지

## ✓ 정적 웹 페이지 (Static Web Page)

- 컴퓨터에 저장된 파일을 그대로 사용자에게 전달하는 웹 페이지
- 모든 상황에서 모든 사용자에게 동일한 정보를 표시
- HTML, CSS, Javascript, 이미지, 동영상 등으로 구성

## ✓ 동적 웹 페이지 (Dynamic Web Page)

- 저장된 데이터를 가공 처리하여 사용자에게 전달하는 웹 페이지
- 상황에 따라 다른 정보를 표시
- 서버 사이드 동적 페이지(Server-side Dynamic Web Page)라고 하며 애플리케이션 서버에 의해 통제
- CGI, PHP, ASP, JSP/Servlet, Node.js



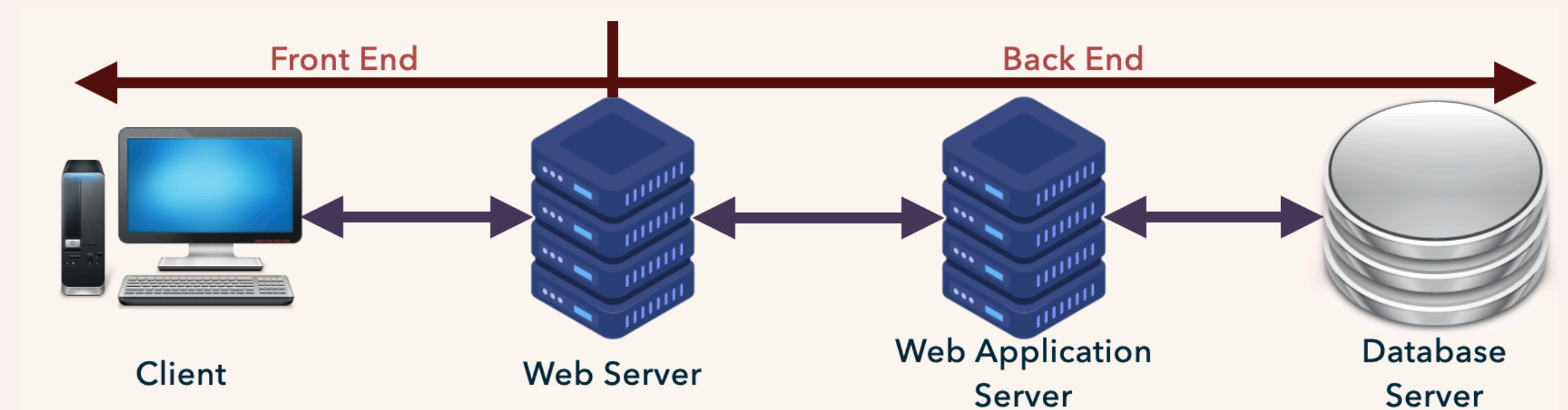
# FRONT END & BACK END

## ✓ 프론트 엔드 (Front End)

- 사용자에게 보여지는 개발 영역으로 결과물을 웹 브라우저를 통해 직접 확인할 수 있음
- **HTML, CSS, Javascript**를 이용하여 개발
- 관련 프레임워크 - React, Vue.js, Angular, JQuery

## ✓ 백 엔드 (Back End)

- 동적 데이터를 처리하는 영역
- JSP/Servlet, PHP, Python, Node.js 등을 이용하여 개발
- 관련 프레임워크 - Spring, Laravel, Django, Express.js



# 백 엔드(BACK END) 기술

## ✓ JSP/Servlet

- Java EE의 기술들로 (JSR 369-servlet, JSR 245-Java Server Page) Java를 기반으로 하여 동적인 콘텐츠를 생성하는 방법을 제공
- 일반적으로 비즈니스 로직을 처리하기 위해 Servlet을 사용하고, 동적 페이지를 작성하기 위해 JSP를 사용

## ✓ Spring 프레임워크

- 자바 플랫폼을 위한 오픈 소스 애플리케이션 프레임워크
- 동적인 웹 사이트를 개발하기 위한 여러 가지 서비스를 제공
- 대한민국 공공기관의 웹 서비스 개발 시 사용을 권장하고 있는 전자정보 표준 프레임워크의 기반 기술



# 백 엔드(BACK END) 기술

## ✓ PHP

- 동적 웹 페이지를 만들기 위해 설계된 프로그래밍 언어
- HTML 문서 안에 PHP로 작성된 코드를 삽입하는 형태
- 범용 프로그래밍 언어로도 사용 가능
- 일반적으로 아파치 웹 서버에서 구동되지만 최근 php-fpm을 통해 실행하는 경우가 늘어나고 있음

## ✓ Laravel 프레임워크

- 오픈 소스 웹 프레임워크의 하나로, MVC 아키텍처 패턴으로 웹 애플리케이션 개발을 위해 고안
- 모듈 방식의 패키징, 의존성 관리, RDBMS 접속, 유지보수를 위한 유틸리티 등을 포함



# 백 엔드(BACK END) 기술

## ✓ Node.js

- 자바스크립트를 활용하여 확장성 있는 네트워크 애플리케이션 (특히 서버 사이드) 개발에 사용 되는 소프트웨어 플랫폼
- Non-blocking I/O와 단일 스레드 이벤트 루프를 통한 높은 처리 성능이 특징

## ✓ Express.js

- Node.js를 위한 오픈 소스 웹 프레임워크
- 웹 애플리케이션, API 개발을 위해 설계
- Node.js의 표준 서버 프레임워크로 불리고 있음
- 폭스 스포츠, 우버, IBM을 포함한 수많은 기업들이 사용



# 백 엔드(BACK END) 기술

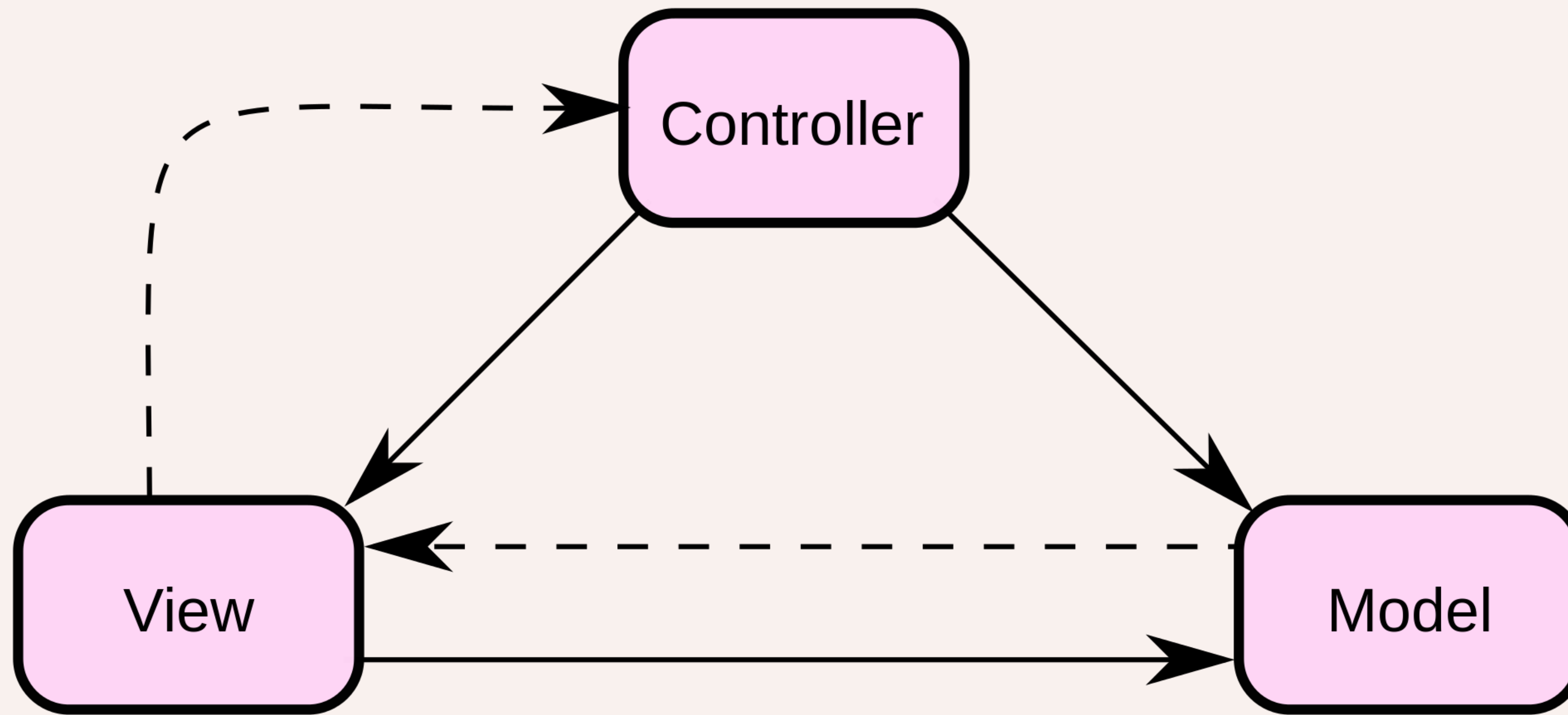
## ✓ Python

- 1991년 귀도 반 로섬이 발표한 고급 프로그래밍 언어
- 플랫폼에 독립적인 객체지향적, 동적 타이핑, 인터프리터 언어
- 라이브러리가 매우 풍부하여 교육기관, 연구기관, 산업계에서 자주 사용

## ✓ Django 프레임워크

- 파이썬으로 작성된 오픈 소스 웹 프레임워크로
- MVC 아키텍처 패턴으로 웹 애플리케이션을 작성하기 위해 고안
- DBMS 기반 웹사이트를 작성하는데 있어 수고를 덜기 위한 것이 주된 목표 (ORM)





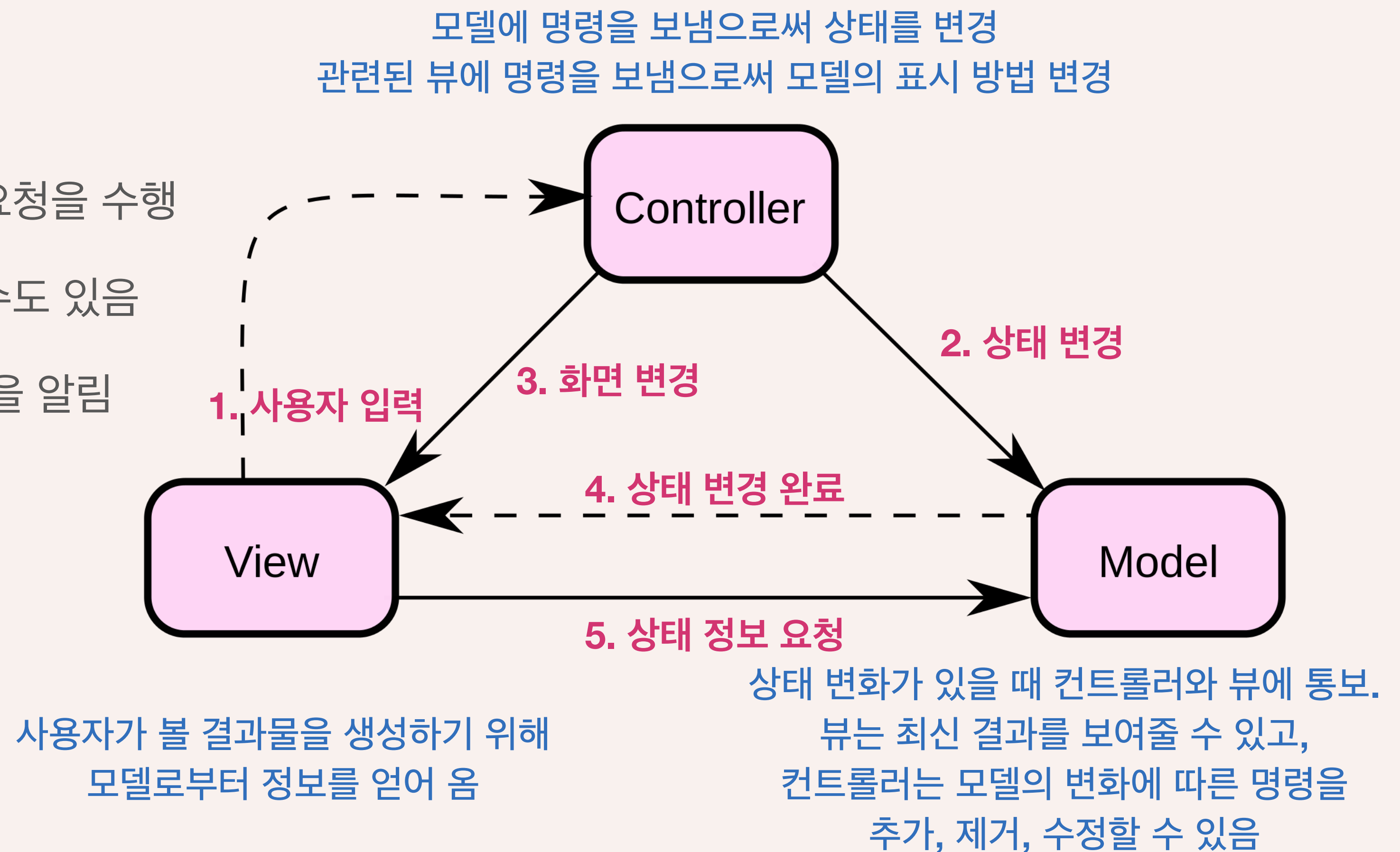
MVC 또는 MTV 구조에서의 게임 클라이언트

02

# MVC와 MTV

## ✓ MVC (Model-View-Controller)

- 사용자는 View만 접근할 수 있음
- 컨트롤러는 모델에게 상태를 변경하라는 요청을 수행
- 컨트롤러에서 뷰를 변경해달라고 요청할 수도 있음
- 상태가 변경되면 모델에서 뷰에게 그 사실을 알림
- 뷰에서 모델에게 상태를 요청

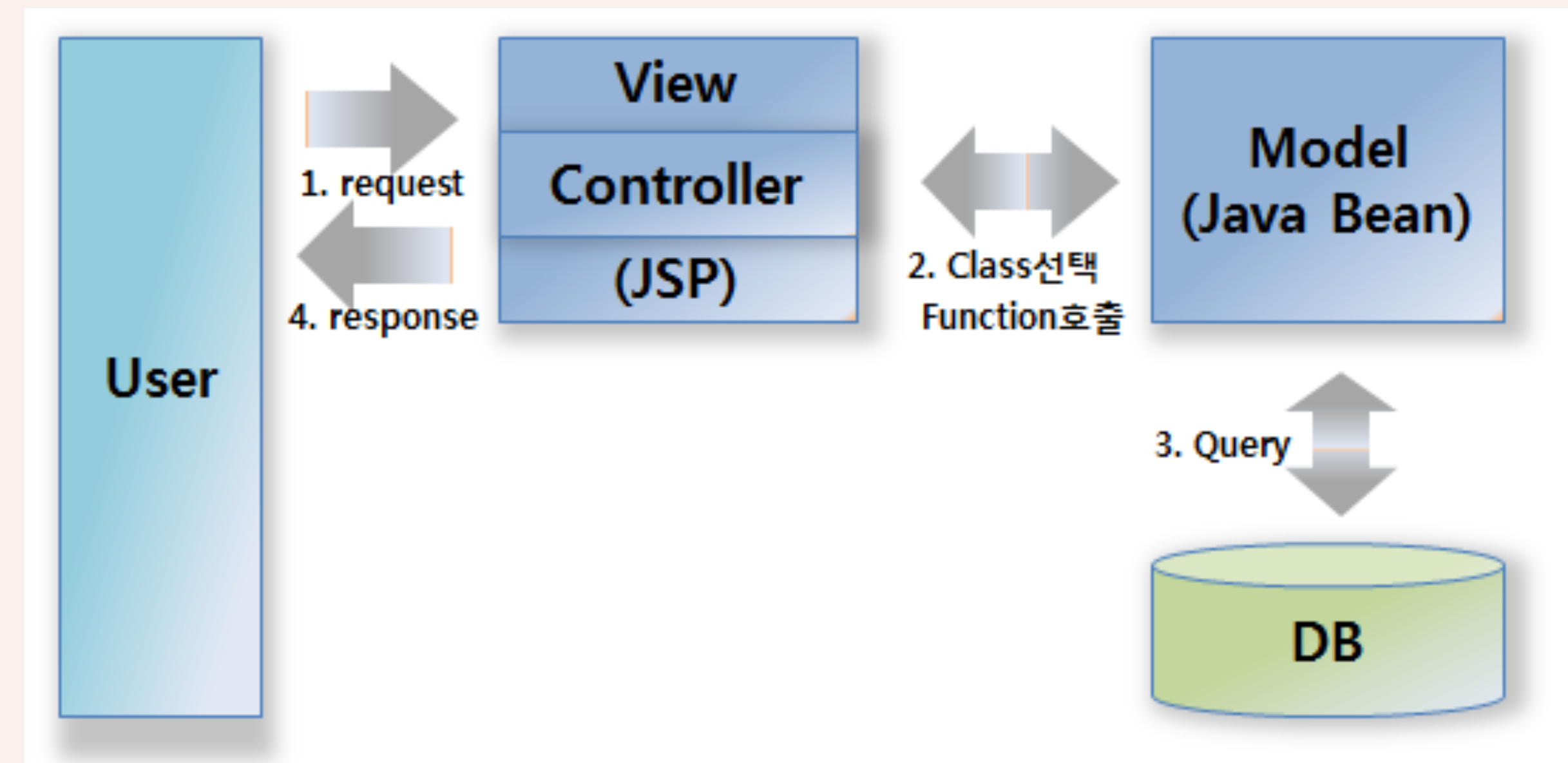




# MVC와 MTV

## ✓ 웹에서의 MVC - MVC1

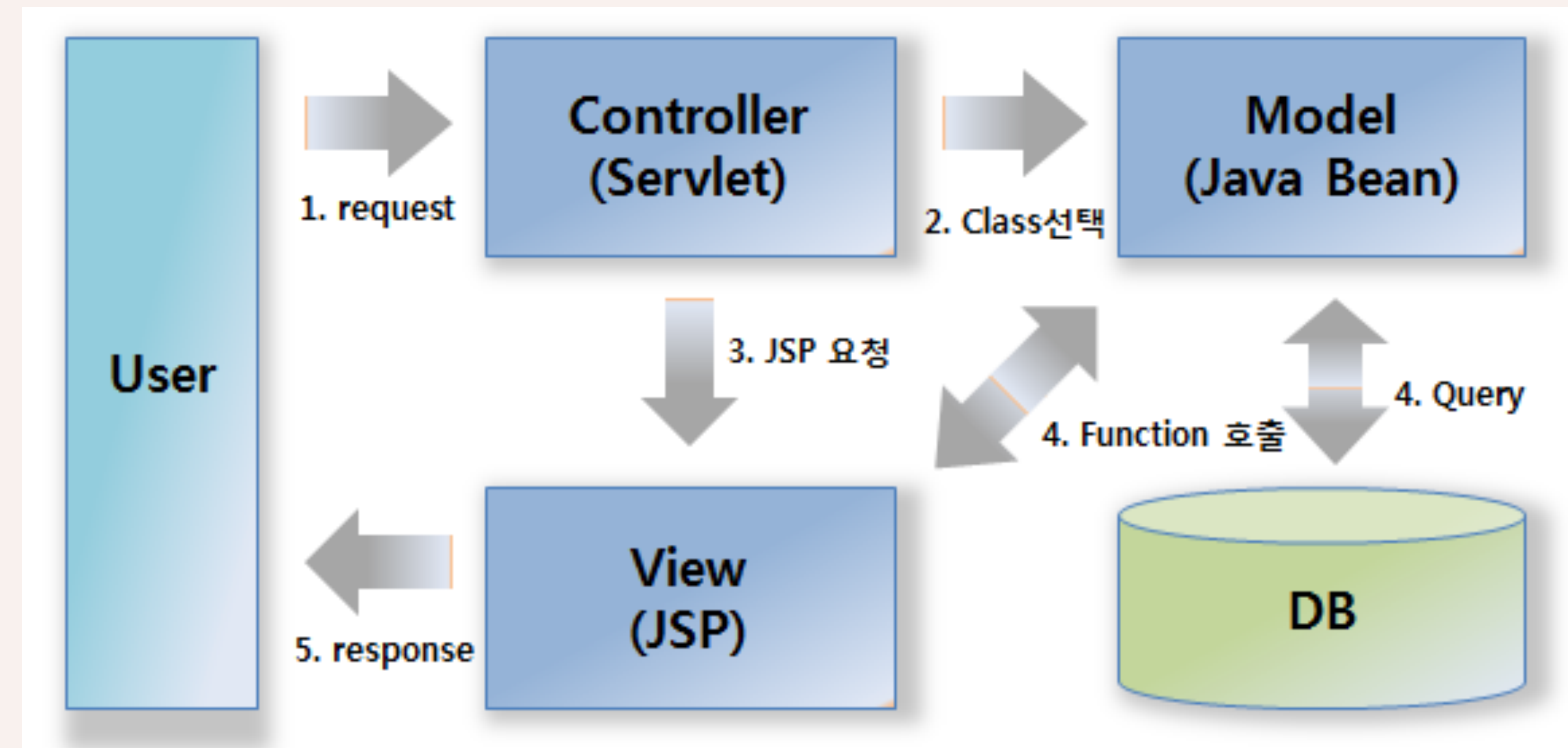
- 단순한 페이지 흐름으로 인해 개발 기간 단축
- MVC 구조에 대한 이해 불필요
- 중소형 프로젝트에 적합
- 단점
  - 웹 어플리케이션이 복잡해 질수록 유지 보수가 힘들
  - 디자이너와 개발자 간의 원활한 의사소통 필요



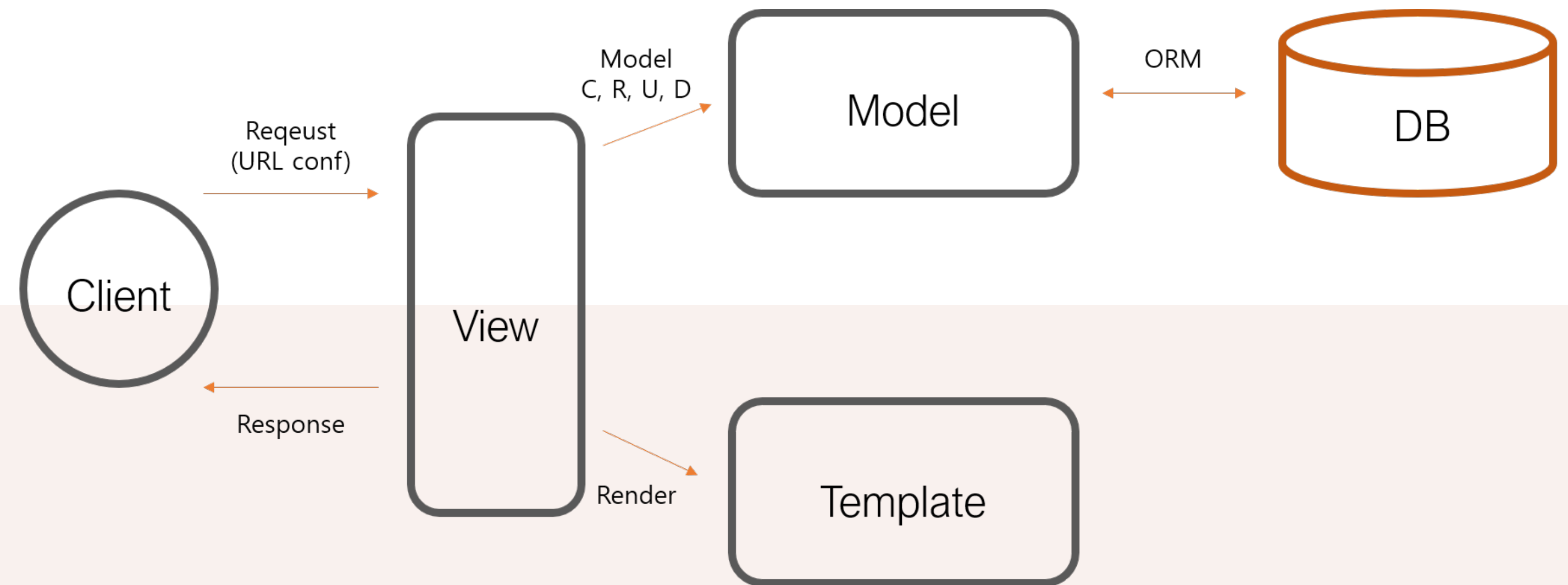
# MVC와 MTV

## ✓ 웹에서의 MVC - MVC2

- 비즈니스 로직(Controller+Model)과 뷰(JSP)의 분리로 인해 어플리케이션이 명료해지며 유지보수와 확장이 용이
- 개발자와 디자이너의 작업이 분리되어서 역할과 책임 구분이 명확해짐
- 단점
  - 구조 설계를 위한 시간이 많이 소요되므로 개발 기간 증가
  - MVC 구조에 대한 이해 필요



# MVC와 MTV



## ✓ MTV (Model-Template-View)

- Model

- 데이터에 관한 정보를 담고 있으며, 데이터에 대한 접근, 검증, 작동과 데이터 사이의 관계를 정의
- 일반적으로 각각의 모델은 데이터베이스에서 테이블에 해당

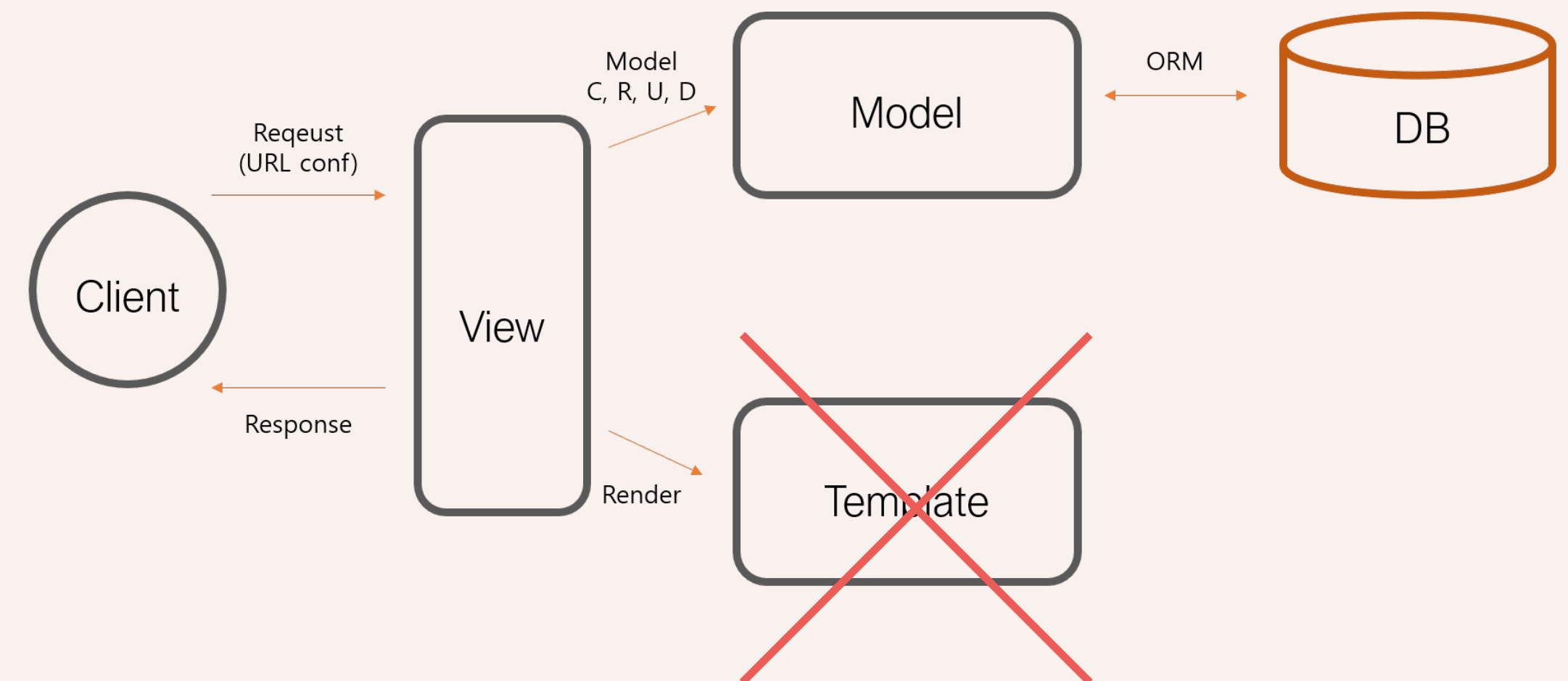
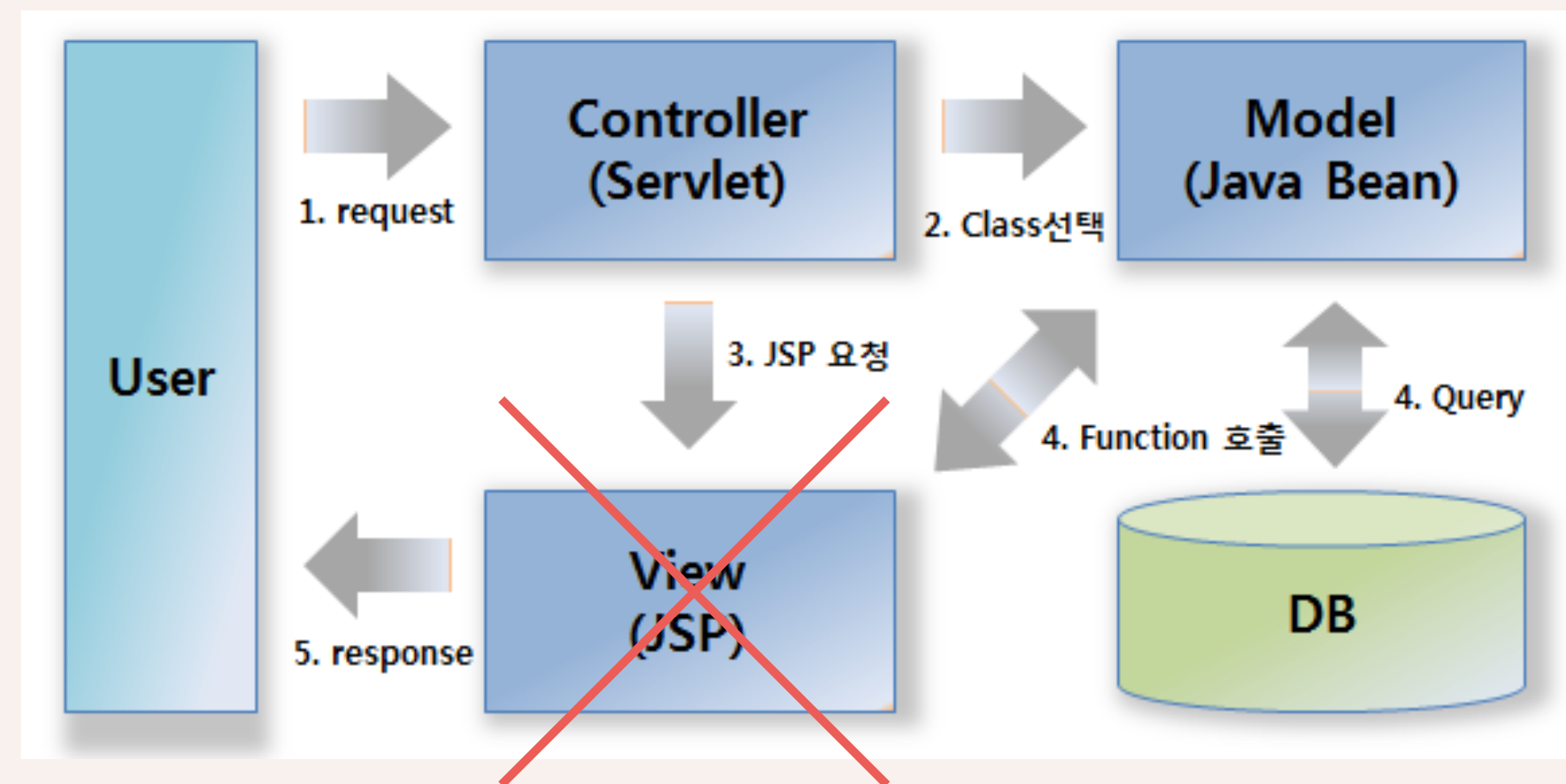
- View

- 어떤 데이터가 표시될 것인지를 정의
- HTTP 응답을 반환해야 하며, 응답의 종류는 웹 페이지, 리다이렉션, 문서, **JSON** 등 다양한 형태가 가능

- Template

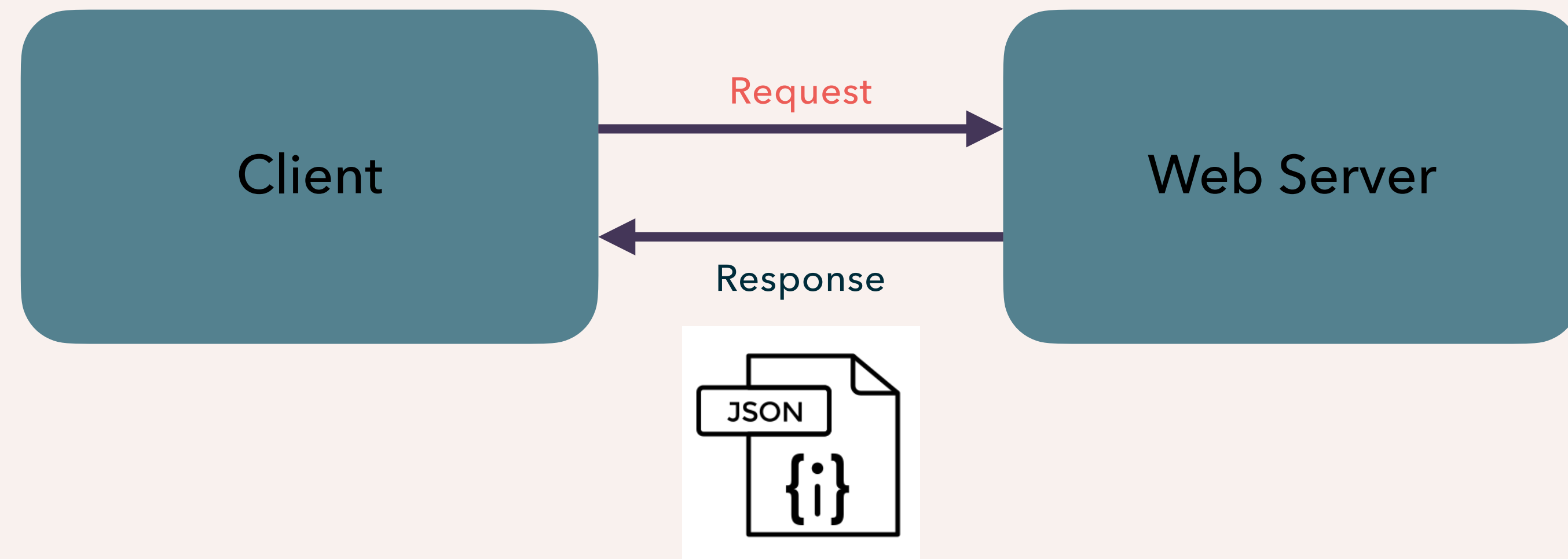
- 데이터가 어떻게 표시되는 지를 정의
- 사용자에게 실제로 보여지는 웹 페이지나 문서를 다룸

# MVC와 MTV



# UNITY와 웹 서버 통신

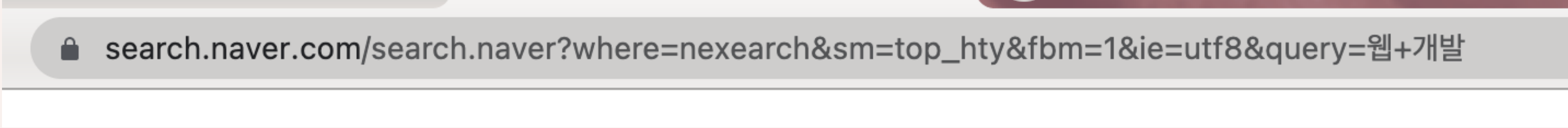
- ✓ 게임 클라이언트가 특정 페이지를 GET/POST 방법으로 웹 서버에 **URL 요청**(request)
- ✓ 웹 서버가 요청을 처리한 후 그 결과를 클라이언트에게 HTML문서, 파일, 문자열 등으로 응답(response)



# UNITY에서 서버로 요청하기

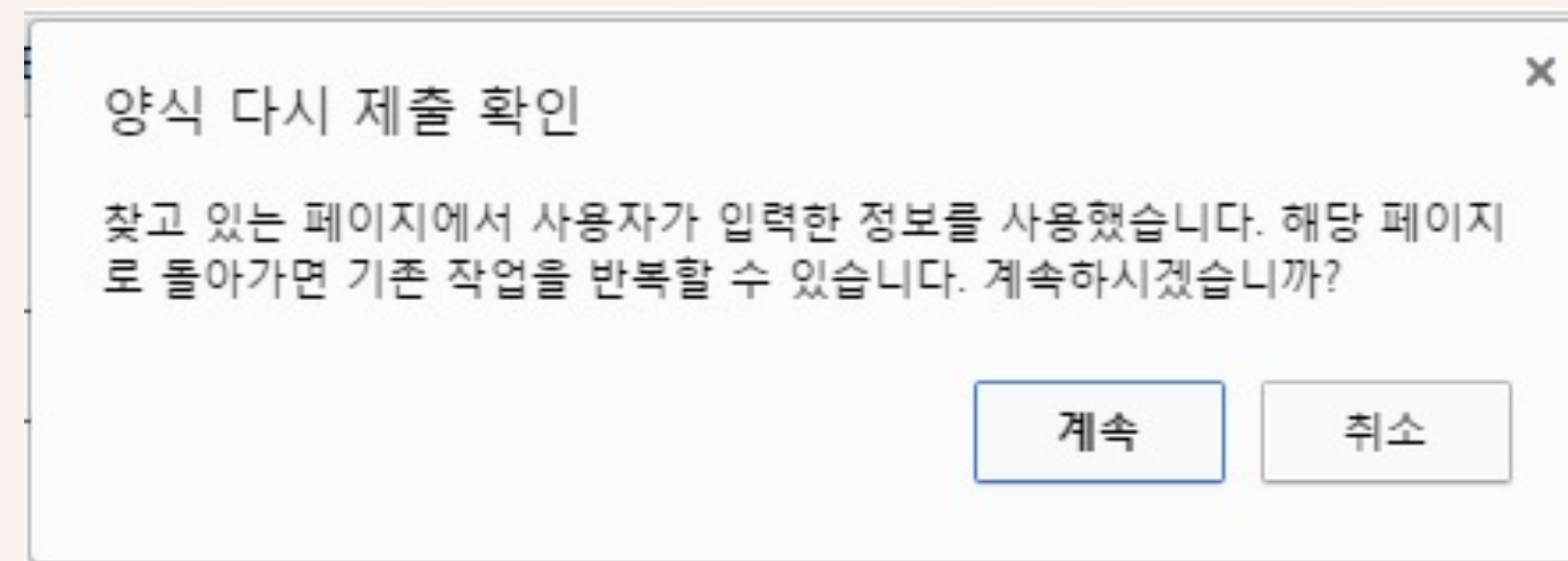
## ✓ 요청 방식

- GET : 데이터를 256~4096 바이트 까지만 서버로 전송가능. 간단하게 데이터를 전송할 수 있으나 주소 표시줄에 사용자가 입력한 내용이 URL에 그대로 드러나 보안상 취약점이 존재하므로 중요한 데이터를 전송하는데 적합하지 않음



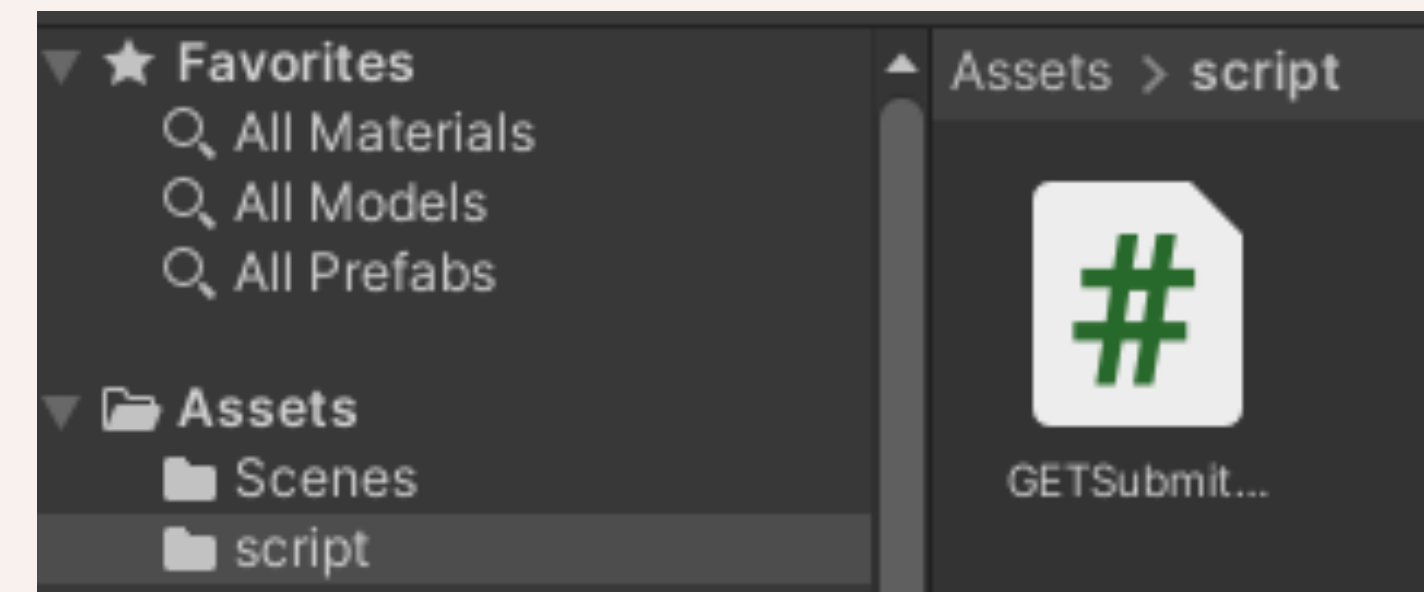
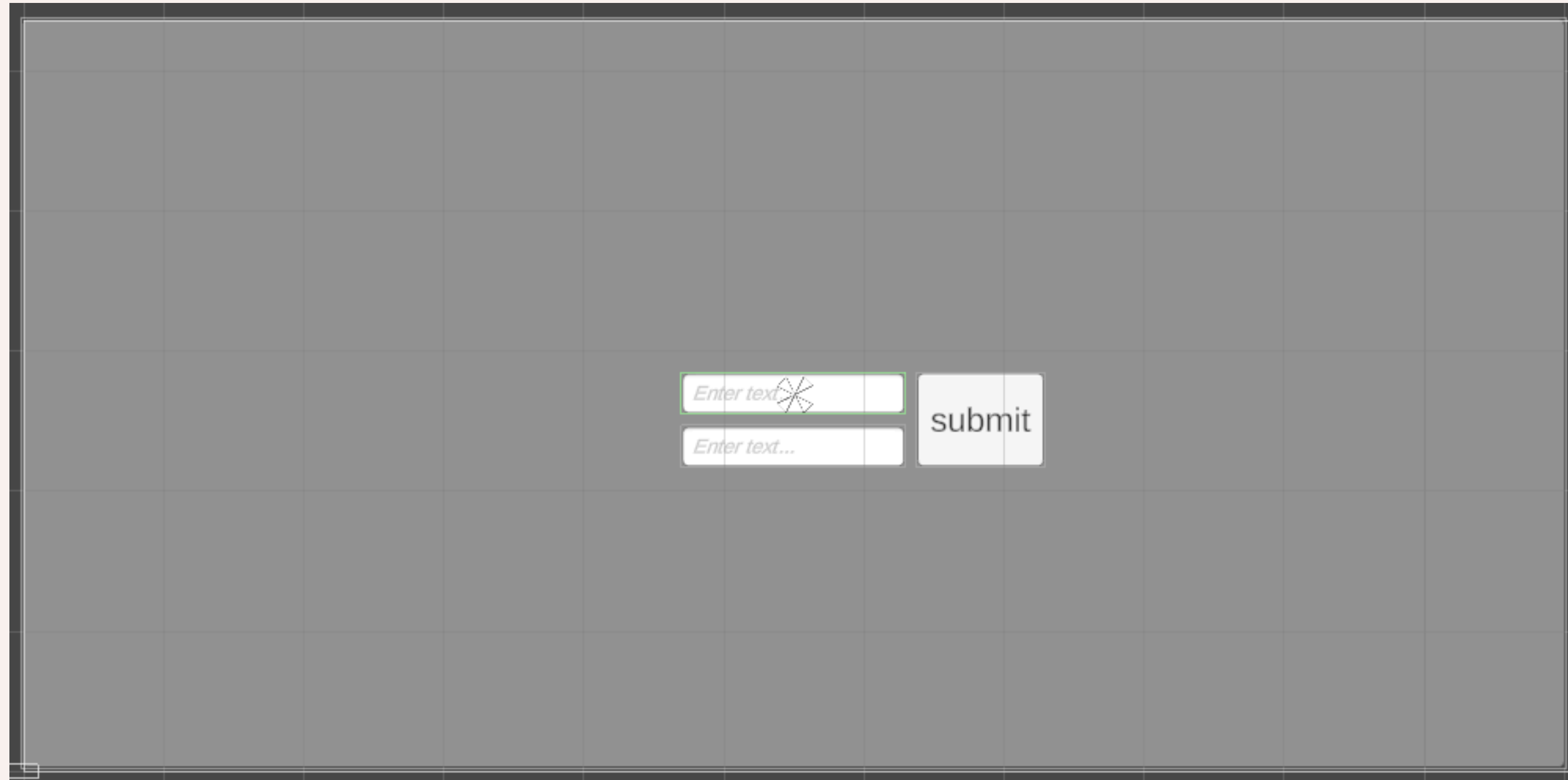
search.naver.com/search.naver?where=nexearch&sm=top\_h ty&fbm=1&ie=utf8&query=웹+개발

- POST : 입력한 내용의 길이에 제한받지 않고 사용자가 입력한 내용이 URL에 드러나지 않으며, 브라우저 히스토리에도 저장되지 않음



# UNITY에서 서버로 요청하기

✓ 준비하기





# UNITY에서 서버로 요청하기

✓ MonoBehaviour의 주요 함수

- Start()
  - 게임 플레이가 시작되기 전 (Update 함수가 호출되기 전 Unity에 의해 호출되며 초기화를 수행)
- Update()
  - 게임 오브젝트에 대한 프레임 업데이트를 처리할 코드를 넣을 곳
  - 움직임, 트리거링 액션, 사용자 입력에 응답하는 기능 등을 포함

```
public class GETSubmitter : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

# UNITY에서 서버로 요청하기

## ✓ 코루틴 (Coroutine)

- 특정 코드가 반복적으로 실행되기 위해 Update() 함수에 코드를 작성
- Update() 함수의 실행 매커니즘(매 프레임마다 실행)과 관계없이 필요한 순간에만 어떤 로직을 실행함으로써 자원관리를 효율적으로 수행
- 높은 성능을 내는 스크립트 제작 가능, 가독성 높은 코드 작성
- 단일 스레드 안에서 동작
- C# 코루틴 선언 방법

```
IEnumerator functionName(param)
```

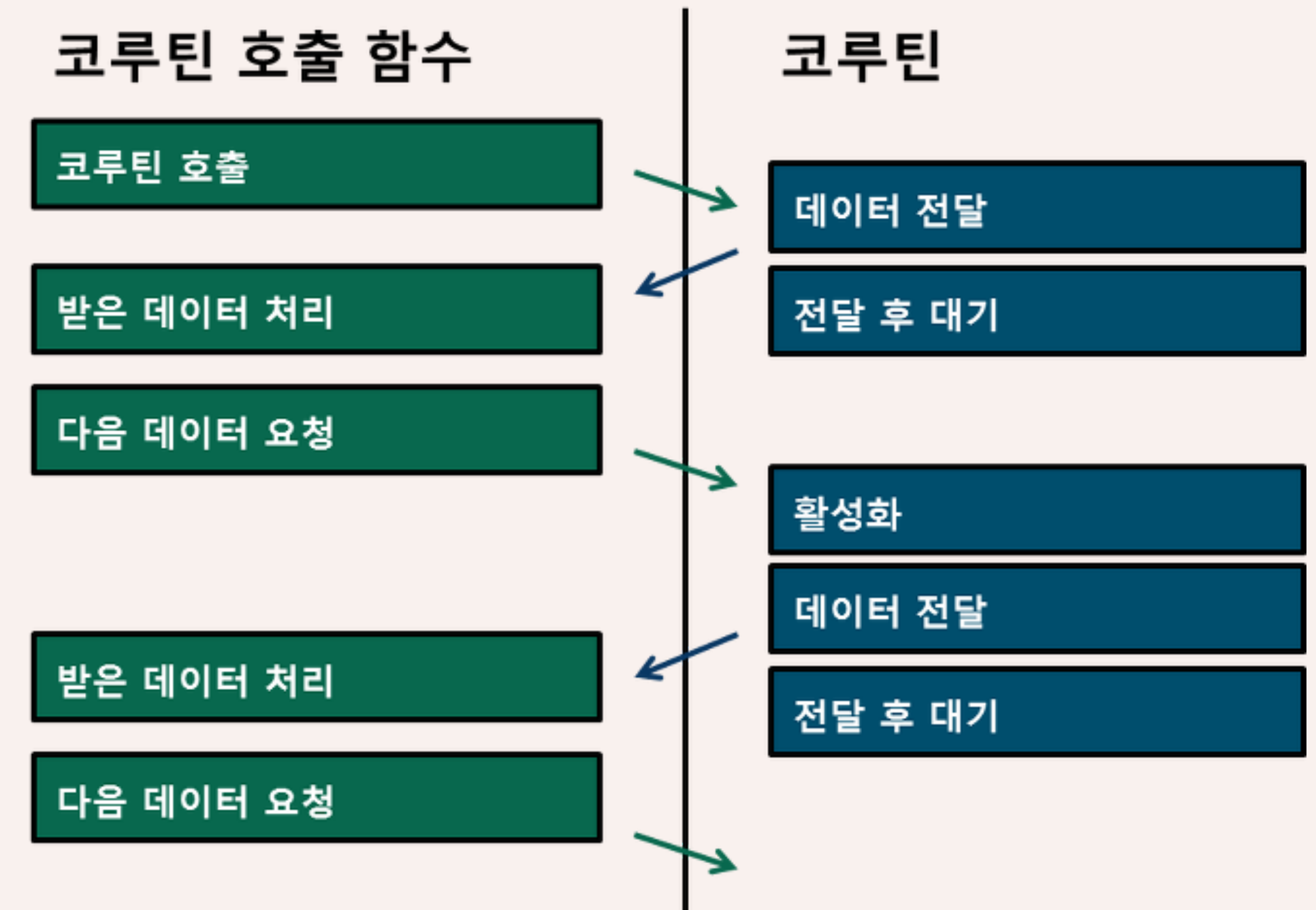
# UNITY에서 서버로 요청하기

## ✓ 코루틴 동작 과정

- 코루틴은 자신을 호출한 함수에 데이터를 넘겨주고 대기
- 코루틴 함수를 호출한 쪽에서 데이터를 받고 처리
- 다음 데이터 요청
- 코루틴은 대기를 해제하여 다시 데이터 전달
- 이 과정을 계속 반복

## ✓ 코루틴 데이터 전달

- 일반적으로 함수를 호출한 쪽으로 데이터를 전달 할 때 return 문 사용
- 코루틴은 데이터를 전달한 후에 자신은 대기 상태로 진입해야 하기 때문에 아래와 같이 사용해야 함



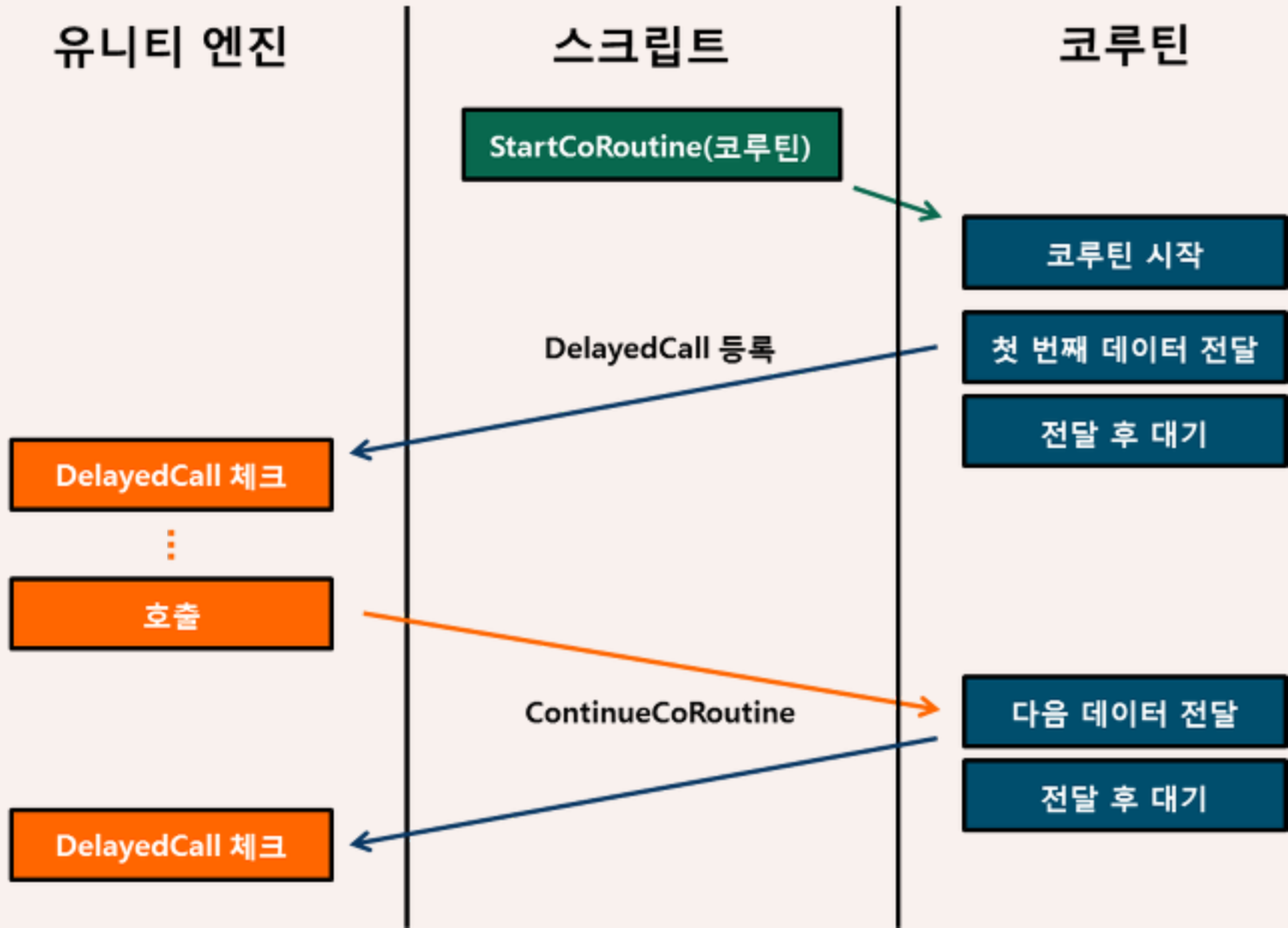
```
yield return ....;
```

# UNITY에서 서버로 요청하기

✓ 코루틴과 Unity 엔진

- 스크립트에서 StartCoroutine() 함수를 사용하여 코루틴 시작
- 유니티 엔진에 DelayedCall 등록 요청
- Delay의 일정 조건을 만족하면 유니티 엔진은 코루틴에 ContinueCoroutine 메시지를 통해 코루틴을 대기 상태에서 해제
- 코루틴이 엔진에 보내는 데이터 종류

코루틴용 데이터	엔진이 수행하는 기능
yield return null	다음 프레임까지 대기
yield return new WaitForSeconds(float)	지정된 초 만큼 대기
yield return new WaitForFixedUpdate()	다음 물리 프레임까지 대기
yield return new WaitForEndOfFrame()	모든 렌더링작업이 끝날 때까지 대기
yield return StartCoroutine(string)	다른 코루틴이 끝날 때까지 대기
yield return return new WWW(string)	웹 통신 작업이 끝날 때까지 대기
yield return new AsyncOperation	비동기 작업이 끝날 때까지 대기(씬로딩)



<https://knightk.tistory.com/3>

# UNITY에서 서버로 요청하기

## ✓ Unity에서 HTTP 통신

- 기존
  - GET 방식 - `UnityEngine WWW`
  - POST 방식 - `UnityEngine WWW`, `UnityEngine WWWForm`
- 최근 Unity
  - `UnityEngine.Networking.UnityWebRequest`

# UNITY에서 서버로 요청하기

- ✓ Unity HTTP GET 방식으로 통신하기 - 씬 로딩 시 통신해야 하는 경우
  - StartCoroutine() - 코루틴 함수 등록
  - IEnumerator 함수() - 코루틴 함수
  - UnityWebRequest - HTTP 통신을 위한 클래스
    - UnityEngine.Networking 임포트 필요
  - yield return www.SendWebRequest() - 유니티 엔진에 데이터 전송

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class GETSubmitter : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(requestServer());
    }

    IEnumerator requestServer() {
        UnityWebRequest www = UnityWebRequest.Get("http://127.0.0.1:8000/main/request1/");

        yield return www.SendWebRequest();

        if (www.error == null) {
            Debug.Log(www.downloadHandler.text);
        } else {
            Debug.Log("Error!!");
        }
    }

    // Update is called once per frame
    void Update()
    {
    }
}
```

# UNITY에서 서버로 요청하기

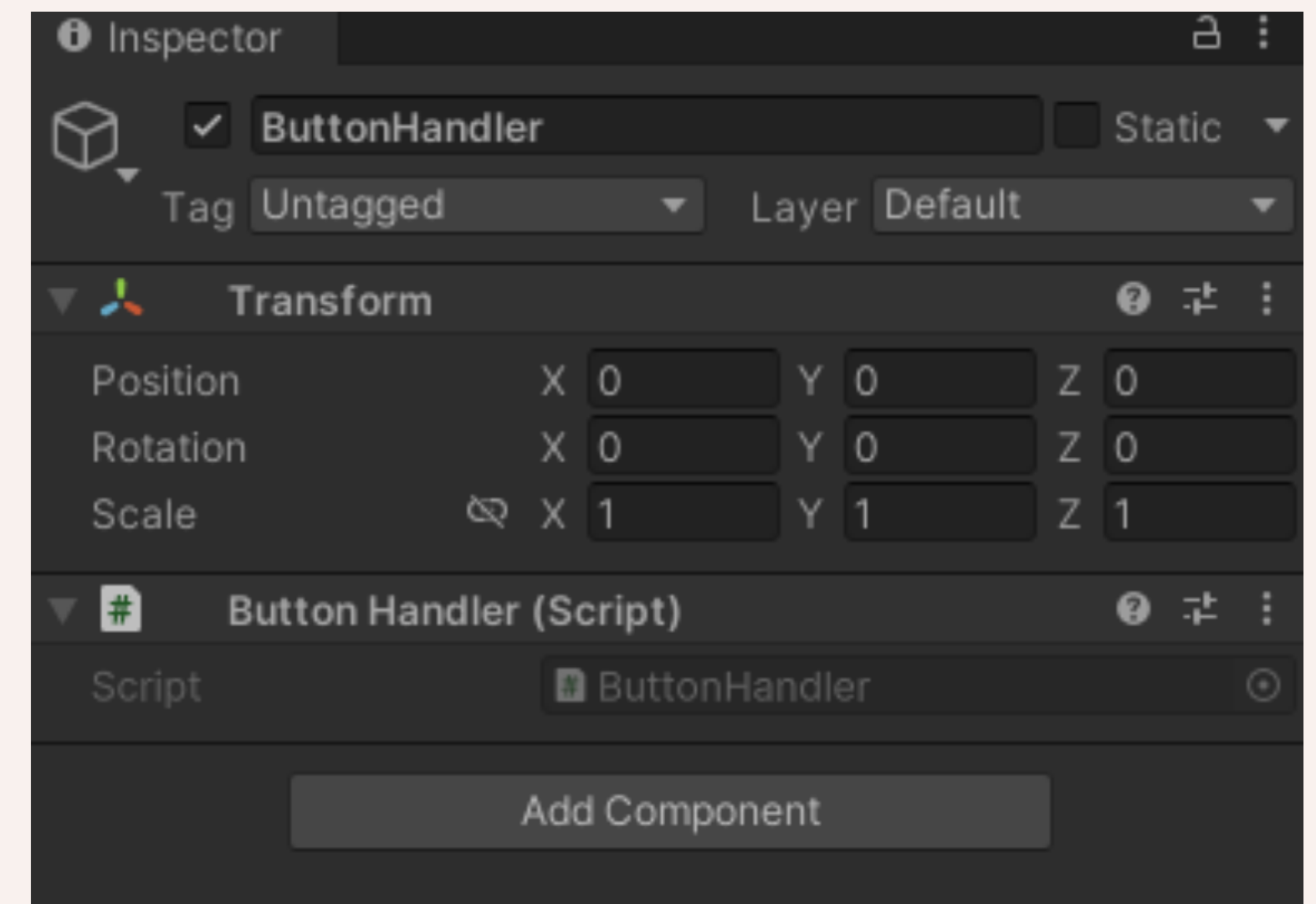
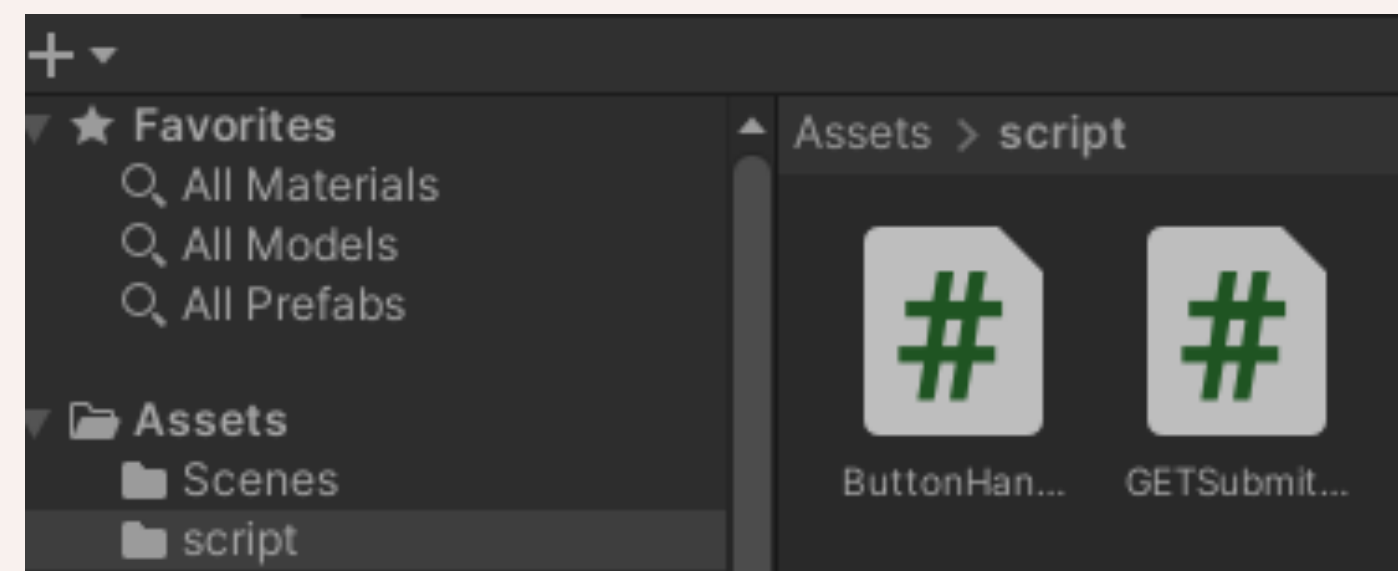
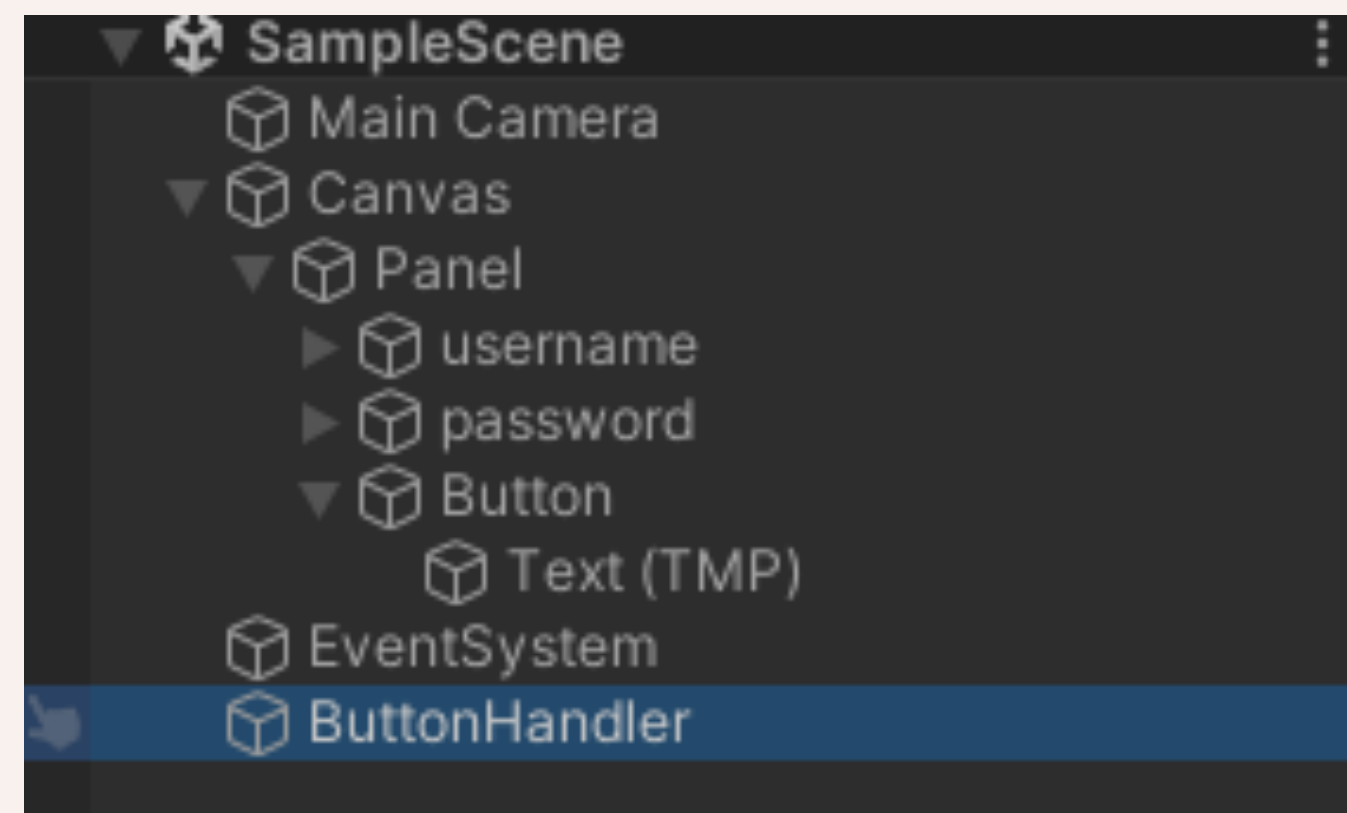
✓ Unity HTTP GET 방식으로 통신하기 - 특정 프레임에서 통신해야 하는 경우

```
IEnumerator requestServer() {  
    UnityWebRequest www = UnityWebRequest.Get("http://127.0.0.1:8000/main/request1/");  
  
    yield return www.SendWebRequest();  
  
    if (www.error == null) {  
        Debug.Log(www.downloadHandler.text);  
    } else {  
        Debug.Log("Error!!");  
    }  
}  
  
// Update is called once per frame  
void Update()  
{  
    // 조건문 필요  
    StartCoroutine(requestServer());  
}
```



# UNITY에서 서버로 요청하기

✓ Unity HTTP GET 방식으로 통신하기 - UI요소 이벤트를 통해 통신하는 경우



# UNITY에서 서버로 요청하기

✓ Unity HTTP GET 방식으로 통신하기 - UI요소 이벤트를 통해 통신하는 경우

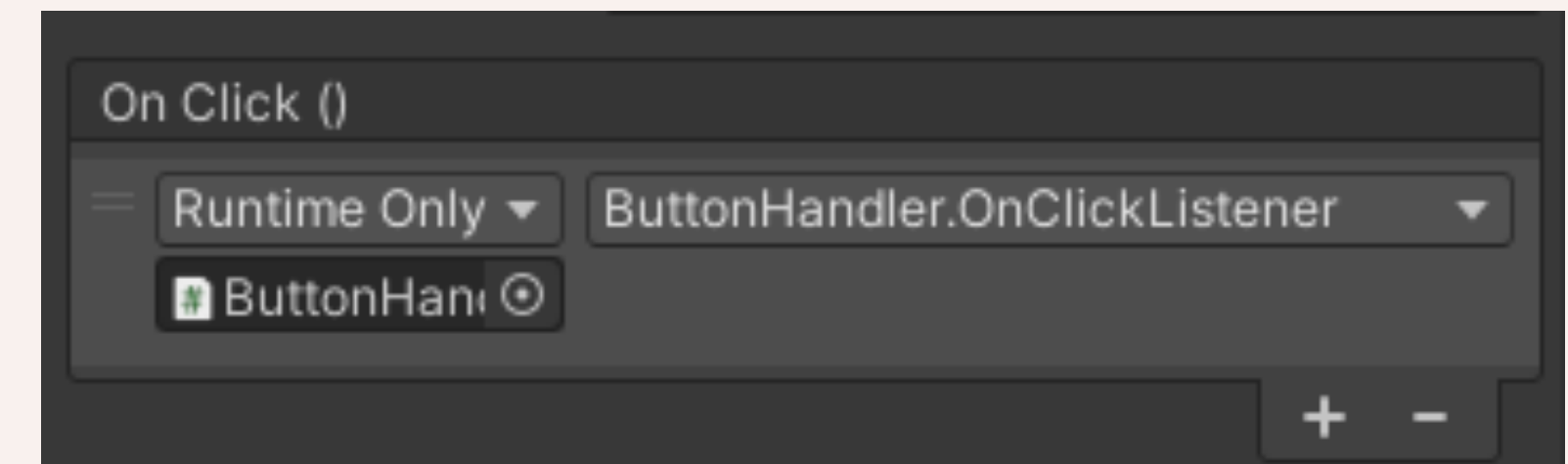
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class ButtonHandler : MonoBehaviour
{
    public void OnClickListener() {
        StartCoroutine(requestServer());
    }

    IEnumerator requestServer() {
        UnityWebRequest www = UnityWebRequest.Get("http://127.0.0.1:8000/main/request1/");

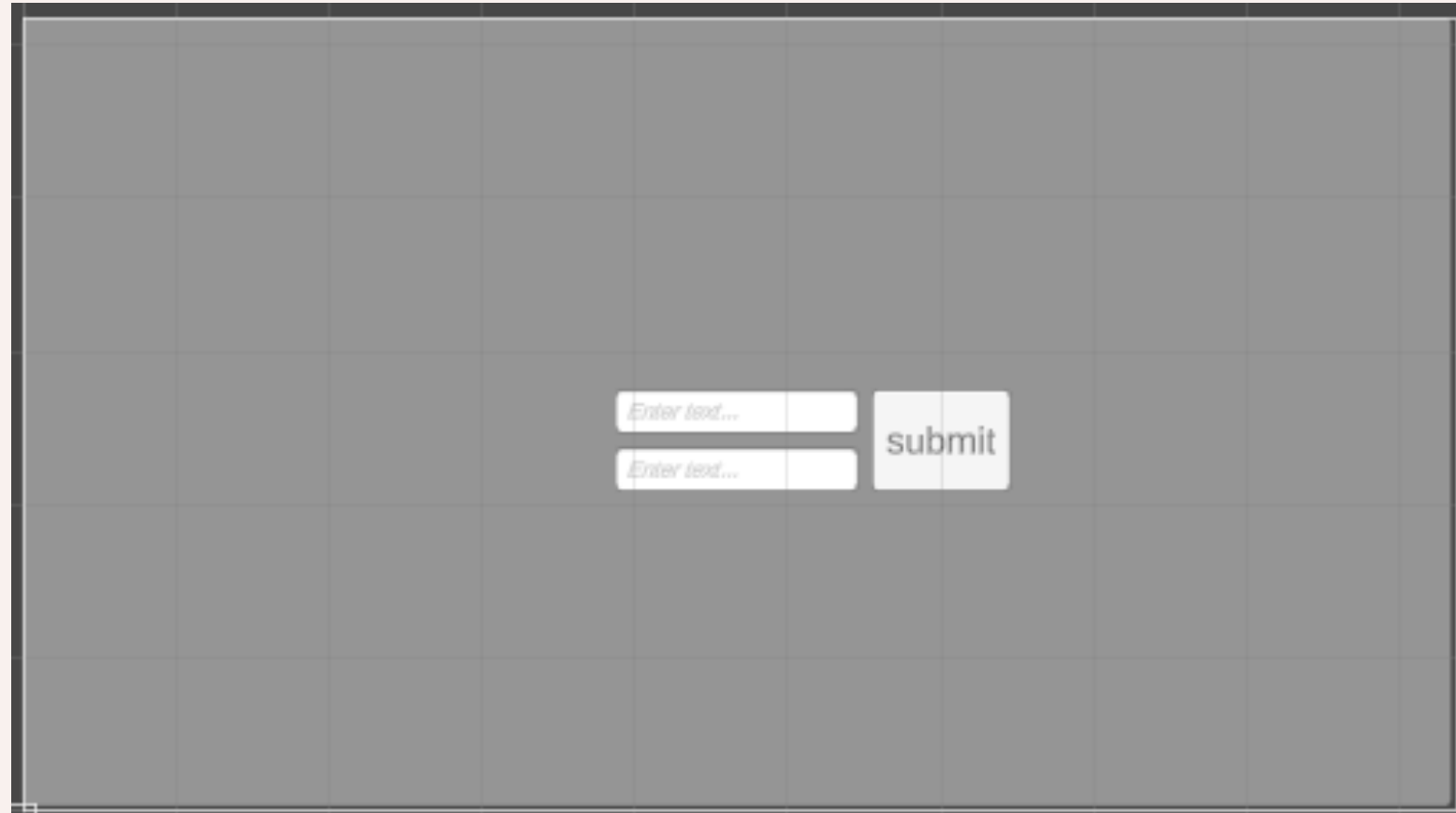
        yield return www.SendWebRequest();

        if (www.error == null) {
            Debug.Log(www.downloadHandler.text);
        } else {
            Debug.Log("Error!!");
        }
    }
}
```



# UNITY에서 서버로 요청하기

✓ Unity HTTP GET 방식으로 통신하기 - 데이터 전송하기



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using TMPro;

public class ButtonHandler : MonoBehaviour
{
    public void OnClickListener() {
        StartCoroutine(requestServer());
    }

    IEnumerator requestServer() {
        GameObject usernameObj = GameObject.Find("username");
        GameObject passwordObj = GameObject.Find("password");

        string username = usernameObj.GetComponent<TMP_InputField>().text;
        string password = passwordObj.GetComponent<TMP_InputField>().text;

        UnityWebRequest www = UnityWebRequest.Get("http://127.0.0.1:8000/main/request1/?username="+username+"&password="+password);

        yield return www.SendWebRequest();

        if (www.error == null) {
            Debug.Log(www.downloadHandler.text);
        } else {
            Debug.Log("Error!!");
        }
    }
}
```

# UNITY에서 서버로 요청하기

- ✓ Unity HTTP POST 방식으로 통신하기 - 데이터 전송하기

```
public class ButtonHandler : MonoBehaviour
{
    public void OnClickListener() {
        StartCoroutine(requestServer());
    }

    IEnumerator requestServer() {
        GameObject usernameObj = GameObject.Find("username");
        GameObject passwordObj = GameObject.Find("password");

        string username = usernameObj.GetComponent<TMP_InputField>().text;
        string password = passwordObj.GetComponent<TMP_InputField>().text;

        WWWForm form = new WWWForm();
        form.AddField("username", username);
        form.AddField("password", password);

        // 파일 전송
        // string path = "경로";
        // byte[] file = System.IO.File.ReadAllBytes(path);
        // form.AddBinaryData("파라미터 이름", file, "저장할 파일 이름", "미디어 타입");

        UnityWebRequest www = UnityWebRequest.Post("http://127.0.0.1:8000/main/request1/", form);

        yield return www.SendWebRequest();

        if (www.error == null) {
            Debug.Log(www.downloadHandler.text);
        } else {
            Debug.Log("Error!!");
        }
    }
}
```

# UNITY에서 서버로 요청하기

## ✓ 미디어 타입 (media type)

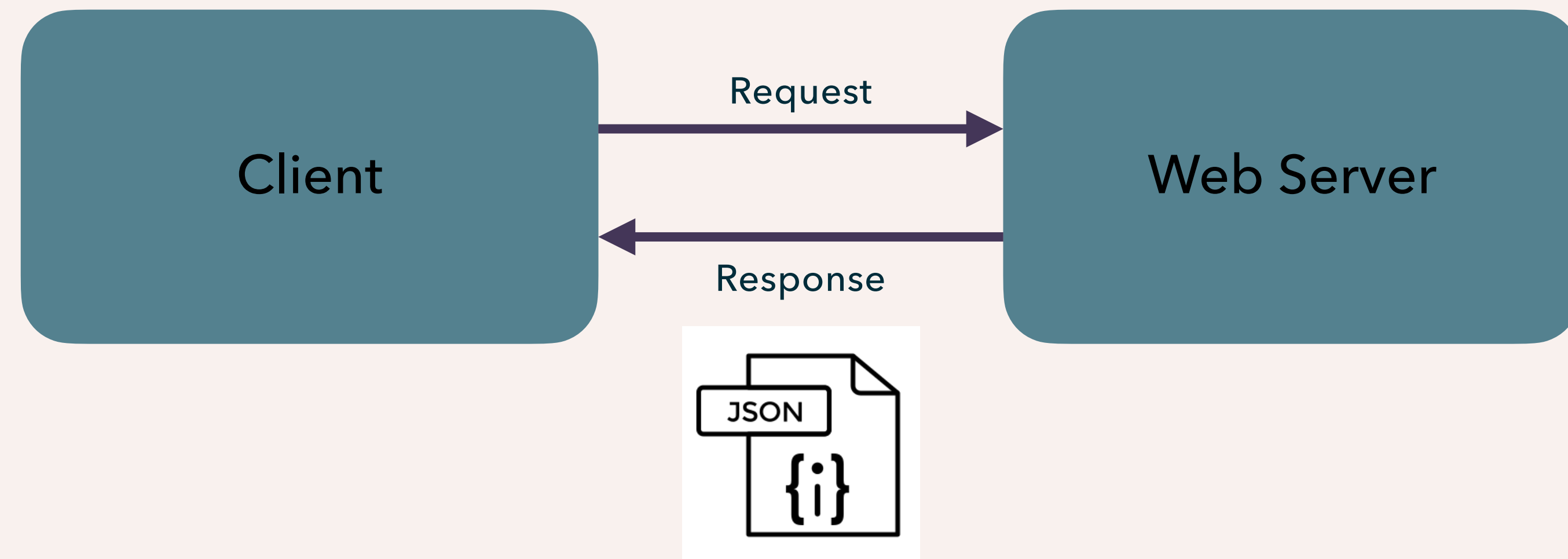
- MIME 타입, 콘텐츠 타입이라고도 하며 인터넷에 전달되는 파일 포맷과 포맷 콘텐츠를 위한 식별자
- type, subtype, 선택적 매개변수로 구성

- application/javascript
- application/json
- application/x-www-form-urlencoded
- application/xml
- application/zip
- application/pdf
- application/sql
- application/graphql
- application/ld+json
- application/msword (.doc)
- application/vnd.openxmlformats-officedocument.wordprocessingml.document (.docx)
- application/vnd.ms-excel (.xls)
- application/vnd.openxmlformats-officedocument.spreadsheetml.sheet (.xlsx)
- application/vnd.ms-powerpoint (.ppt)
- application/vnd.openxmlformats-officedocument.presentationml.presentation (.pptx)
- application/vnd.oasis.opendocument.text (.odt)
- audio/mpeg
- audio/vorbis
- multipart/form-data
- text/css
- text/html
- text/csv
- text/plain
- image/png
- image/jpeg
- image/gif



# UNITY와 웹 서버 통신

- ✓ 게임 클라이언트가 특정 페이지를 GET/POST 방법으로 웹 서버에 URL 요청(request)
- ✓ 웹 서버가 요청을 처리한 후 그 결과를 클라이언트에게 HTML문서, 파일, 문자열 등으로 응답(response)



# JSON

## ✓ JSON (JavaScript Object Notation)

- “키-값”의 쌍으로 이루어진 데이터 객체를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개발형 표준 포맷
- 좁게는 비동기 브라우저/서버 통신 (AJAX)에 사용
- 넓게는 XML을 대체하는 주요한 데이터 포맷으로 사용
- 컴퓨터 프로그램의 변수값을 표현하는데 매우 적합한 포맷
- 자바스크립트로부터 파생되었지만 언어 독립형 데이터 포맷
- 주석 사용을 권장하지 않음

```
[
  {
    "description": "quarter",
    "mode": "REQUIRED",
    "name": "qtr",
    "type": "STRING"
  },
  {
    "description": "sales representative",
    "mode": "NULLABLE",
    "name": "rep",
    "type": "STRING"
  },
  {
    "description": "total sales",
    "mode": "NULLABLE",
    "name": "sales",
    "type": "INTEGER"
  }
]
```



# JSON

## ✓ JSON의 기본 자료형

- Number
- String
- Boolean
- Array
- Object
- null

# JSON

## ✓ Number

- 기본 자료형 Number는 다음과 같이 표현
- 10진수 체계가 기본이며 8진수, 16진수를 표현하는 방법을 지원하지 않음

### • 정수

74  
1974  
750  
-114  
-273

### • 실수(고정 소수점)

3.14  
-2.718

### • 실수(부동소수점)

1e4  
2.5e12  
3.4e+4  
4.56e-8  
5.67E+10  
6.78E-5

# JSON

## ✓ String

- 항상 큰 따옴표로 묶어야 하며, 그 안에는 유니코드 문자들이 나열됨
  - UTF-8 (기본), UTF-16, UTF-32 사용 가능
- 제어문자가 포함될 수 있음

```
"1234"  
"Love"  
"0-matic"  
"한글"  
"\"JSON\""
```

\b 백스페이스

\f 폼 피드

\n 개행

\r 캐리지 리턴

\t 탭

\" 따옴표

\\ 슬래시

\\ 역슬래시

\uHHHH 16진수 네자리로되어 있는 유니코드 문자

# JSON

## ✓ Array

- 대괄호 [ ] 를 이용하여 작성
- 배열의 각 요소는 기본 자료형이거나 숫자, 문자열, , null, Boolean, 다른 배열, 객체 데이터 등으로 구성될 수 있음
- 쉼표로 구분되며 순서가 있음

```
[10, {"v": 20}, [30, "마흔"]]
```

# JSON

## ✓ Object

- 중괄호를 사용하여 객체는 이름/값 쌍의 집합으로 표현
- 여러개의 데이터가 있을 경우 쉼표로 구분
- 이름은 문자열이어야 하며, 값은 JSON 자료형으로 작성될 수 있음
- 순서가 없음

```
{  
  "이름": "홍길동",  
  "나이": 25,  
  "성별": "여",  
  "주소": "서울특별시 양천구 목동",  
  "특기": ["농구", "도술"],  
  "가족관계": {"#": 2, "아버지": "홍판서", "어머니": "춘섬"},  
  "회사": "경기 수원시 팔달구 우만동"  
}
```

# JSON

## ✓ Boolean

- 참/거짓 값을 표현하는 true, false 두 개의 값만 갖는 데이터

## ✓ null

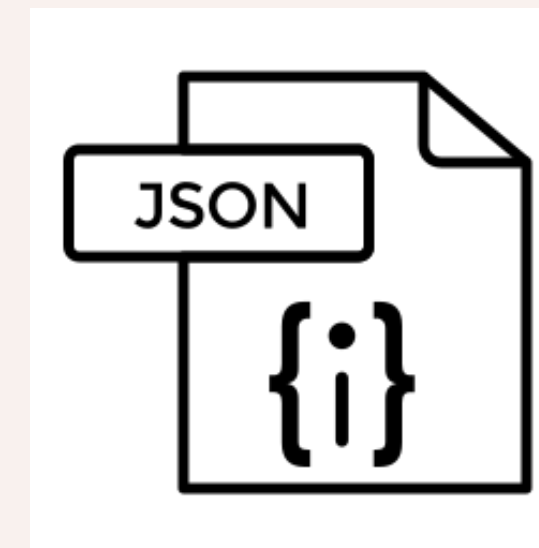
- 빈 값을 의미하는 데이터
- 빈 값은 공백문자, 0을 의미하지 않고, 말 그대로 값이 없다는 것을 의미
- 아무 값도 할당되지 않은 상태를 의미

# JSON

## ✓ 주의

- JSON은 프로그래밍 언어가 아니라 데이터 종류 중 하나이기 때문에 작성 시 오류가 있다고 해도 에러를 발생시키지 않음
- JSON을 읽어들이는 프로그래밍 언어에서 해당 오류를 출력

✓ 오류를 줄이는 가장 좋은 방법은 JSON 모듈을 사용하는 것



오류가 있는  
JSON 파일 또는 문자열

**에러 발생하지 않음**



JSON Module

**에러 발생**



# UNITY에서 JSON 응답받기

## ✓ JsonUtility

- JSON 데이터를 처리하기 위한 여러 유틸리티 기능을 포함
- 모든 메소드가 정적(static) 메소드로 선언되어 있음

static methods	설명
FromJson	JSON 데이터를 표현하는 객체 생성
FromJsonOverwrite	JSON 데이터를 표현하는 객체에 다시 쓰기
ToJson	객체의 public 필드를 표현하는 JSON 데이터 생성

# UNITY에서 JSON 응답받기

## ✓ JsonUtility.FromJson

- 내부적으로 Unity Serializer를 사용하여 JSON 데이터를 객체로 생성
- 직렬화 가능한 속성으로 일반 클래스나 구조체를 생성해야 함
- 객체의 필드에는 Serializer가 지원하는 타입이 있어야 하며, 지원되지 않는 필드와 NonSerialized 속성으로 표시된 필드는 무시
- UnityEngine.Object에서 파생된 클래스는 지원되지 않음 -> 일반 클래스만 지원

# UNITY에서 JSON 응답받기

## ✓ JSON 응답받기 위한 과정 1

- JSON 데이터를 표현하는 직렬화 클래스나 구조체 생성
- 이때 **JSON key값**과 변수명과 **JSON 데이터와 클래스 필드 데이터의 자료형**이 일치하도록 해야 함
- JSON 자료형에 대응되는 C# 자료형

JSON 자료형	C# 자료형	JSON 자료형	C# 자료형
Number	int	Array	배열
String	string	Object	class
Boolean - true, false	bool	null	null

# UNITY에서 JSON 응답받기

✓ JSON 응답받기 위한 과정 1 - JSON 데이터와 대응되는 직렬화 클래스 만들기

```
{
  "result":true,
  "item":
  [
    {"item1":"a","item2":"b"},
    {"item1":"c","item2":"d"}
  ]
}
```

```
[System.Serializable]
class Item
{
    public string item1;
    public string item2;
}

[System.Serializable]
class Data
{
    public bool result;
    public Item[] item;
}
```

# UNITY에서 JSON 응답받기

## ✓ JSON 응답받기 위한 과정 2

- Http로 response된 JSON 데이터 문자열을 JsonUtility.FromJson을 이용하여 파싱
- JsonUtility.FromJson<클래스>("JSON 문자열")

```
IEnumerator requestServer() {  
    UnityWebRequest www = UnityWebRequest.Get("http://127.0.0.1:8000/main/request2/");  
  
    yield return www.SendWebRequest();  
  
    if (www.error == null) {  
        Data data = JsonUtility.FromJson<Data>(www.downloadHandler.text);  
        Debug.Log(data.result);  
        Debug.Log(data.item[0].item1 + " " + data.item[0].item2);  
        Debug.Log(data.item[1].item1 + " " + data.item[1].item2);  
    } else {  
        Debug.Log("Error!!");  
    }  
}
```