

# Section 1. 생애 최초 API 만들기 목표

1. 스프링 프로젝트를 설정해 시작하고 실행할 수 있다.
2. 서버란 무엇인지, 네트워크와 HTTP, API는 무엇인지, JSON은 무엇인지 등 서버 개발에 필요한 다양한 개념을 이해한다.
3. 스프링 부트를 이용해 간단한 GET API, POST API를 만들 수 있다.

# 스프링 프로젝트를 어떻게 시작할 수 있을까?!

1. 이미 만들어져 있는 스프링 프로젝트를 다운로드 받기
2. spring initializr를 이용해 새로운 프로젝트 시작하기

# **[1] 기존의 프로젝트 시작하기**

<수업 소스코드 준비하기>에 영상으로 있습니다!

## [2] 새로운 프로젝트 시작하기

<https://start.spring.io>

# [2] 새로운 프로젝트 시작하기

## Project

☒ Gradle Project ☐ Maven Project

## Language

☒ Java ☐ Kotlin ☐ Groovy

## Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC1) ☐ 2.7.6 (SNAPSHOT) ☒ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

## Project Metadata

Group com.example

Artifact demo

Name demo

Description Demo project for Spring Boot

Package name com.example.demo

Packaging ☒ Jar ☐ War

Java ☐ 19 ☒ 17 ☐ 11 ☐ 8

## [2] 새로운 프로젝트 시작하기

**Project**  
☒ Gradle Project ☐ Maven Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC1) ☐ 2.7.6 (SNAPSHOT) ☒ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

**Project Metadata**  

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for Spring Boot

Package name

com.example.demo

Packaging

☒ Jar ☐ War

Java

☐ 19 ☒ 17 ☐ 11 ☐ 8

이 프로젝트에서 사용될 '빌드 툴'

최근에는 Gradle이 많이 사용된다.

## [2] 새로운 프로젝트 시작하기

**Project**  
☒ Gradle Project ☐ Maven Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC1) ☐ 2.7.6 (SNAPSHOT) ☒ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

**Project Metadata**

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for Spring Boot

Package name

com.example.demo

Packaging

☒ Jar ☐ War

Java

☐ 19 ☒ 17 ☐ 11 ☐ 8

서버를 개발할 때 사용할 언어

최신 프로젝트에서는 Kotlin을 사용하는 경향이 있지만, Java로 만들어진 기존 프로젝트가 많이 존재한다.

## [2] 새로운 프로젝트 시작하기

### Project

☒ Gradle Project ☐ Maven Project

### Language

☒ Java ☐ Kotlin ☐ Groovy

### Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC1) ☐ 2.7.6 (SNAPSHOT) ☒ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

### Project Metadata

Group com.example

Artifact demo

Name demo

Description Demo project for Spring Boot

Package name com.example.demo

Packaging ☒ Jar ☐ War

Java ☐ 19 ☒ 17 ☐ 11 ☐ 8

스프링 부트의 버전을 고르는 항목

알파벳이 붙어 있으면,  
개발 중이거나 오픈 베타라는  
의미이다.

시간이 지나면서 계속 버전이  
업그레이드 되기 때문에,  
강의를 보는 시점에 따라 다른  
숫자가 나올 수 있다.



## [2] 새로운 프로젝트 시작하기

**Project**  
☒ Gradle Project ☐ Maven Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC1) ☐ 2.7.6 (SNAPSHOT) ☒ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

**Project Metadata**

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo

**Packaging** ☒ Jar ☐ War

**Java** ☐ 19 ☒ 17 ☐ 11 ☐ 8

프로젝트에 존재하는  
다양한 이름을 짓는 항목

Group : 프로젝트 그룹

Artifact : 최종 결과물의 이름

Name : 프로젝트 이름

Description : 프로젝트 설명

Package name : 패키지 이름

## [2] 새로운 프로젝트 시작하기

**Project**  
☒ Gradle Project ☐ Maven Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC1) ☐ 2.7.6 (SNAPSHOT) ☒ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

**Project Metadata**  

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for Spring Boot

Package name

com.example.demo

Packaging

☒ Jar ☐ War

Java

☐ 19 ☒ 17 ☐ 11 ☐ 8

Spring Boot는 톰캣이 내장되어 있어 Jar을 선택하면 된다.

이 강의를 통해 톰캣이 무엇인지, 내장되어 있다는 의미가 무엇인지 모두 설명드릴 예정입니다!

## [2] 새로운 프로젝트 시작하기

**Project**  
☒ Gradle Project ☐ Maven Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC1) ☐ 2.7.6 (SNAPSHOT) ☒ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

**Project Metadata**  

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for Spring Boot

Package name

com.example.demo

Packaging

☒ Jar ☐ War

Java

☐ 19 ☒ 17 ☐ 11 ☐ 8

### Java의 버전

기존에 존재하는 프로젝트는  
Java11이 가장 많고,  
그 다음은 Java8이 많다.

최신 프로젝트는 최신 Java 버전을  
사용할 수 있다.

# [2] 새로운 프로젝트 시작하기

## Dependencies

ADD DEPENDENCIES... ⌘ + B

### Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.



### H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.



### Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.



의존성을 설정한다.

# [2] 새로운 프로젝트 시작하기

## Dependencies

ADD DEPENDENCIES... ⌘ + B

### Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.



### H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.



### Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.



의존성이란, 프로젝트에서 사용하는 라이브러리 / 프레임워크를 의미한다.

# 라이브러리란?

프로그래밍을 개발할 때  
미리 만들어져 있는 기능을 가져다 사용하는 것

# 라이브러리란? (요리 비유)

김치찌개를 만들자!

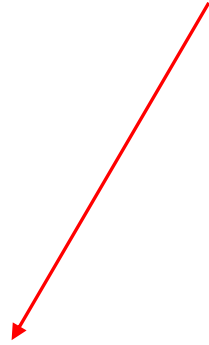
# 라이브러리란? (요리 비유)

김치를 마트에서 사서 만들 수도 있고

배추 농사부터 지을 수도 있다.



# 라이브러리란? (요리 비유)



김치를 마트에서 사서 만들 수도 있고

배추 농사부터 지을 수도 있다.

# 프레임워크?

프로그래밍을 개발할 때  
미리 만들어져 있는 구조에 코드를 가져다 끼워 넣는 것

# 프레임워크? (요리 비유)

김치찌개를 만들자!

# 프레임워크? (요리 비유)

여러 재료를 사서 만들 수도 있고

원데이 클래스에 가서 선생님이 시키는 것만 편하게 할 수도 있다.

# 프레임워크? (요리 비유)

여러 재료를 사서 만들 수도 있고

원데이 클래스에 가서 선생님이 시키는 것만 편하게 할 수도 있다.

# [2] 새로운 프로젝트 시작하기

## Dependencies

ADD DEPENDENCIES... ⌘ + B

### Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.



### H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.



### Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.



각 자료에 대해서는  
이번 강의에서 하나씩 다루게 된다!

# [2] 새로운 프로젝트 시작하기



**Project**  
☒ Gradle Project ☐ Maven Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC1) ☐ 2.7.6 (SNAPSHOT) ☒ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

## Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 19 ☒ 17 ☐ 11 ☐ 8

## Dependencies

ADD DEPENDENCIES... ⌘ + B

No dependency selected

GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

SHARE...

# 2강. @SpringBootApplication과 서버



# 서버란 무엇인가?

Server

# 서버란 무엇인가?

Server = Serve + er (~하는 사람)

# 서버란 무엇인가?

serve



1. 동사 (식당 등에서 음식을) 제공하다, (음식을 상에) 차려 주다[내다]
2. 동사 돌아가다
3. 명사 서브 (널기)

# 서버란 무엇인가?

serve



1. 동사 (식당 등에서 음식을) **제공**하다, (음식을 상에) 차려 주다[내다]
2. 동사 돌아가다
3. 명사 서브 (널기)

# 서버란 무엇인가?

Server = 제공하는 것(사람)

# 서버란 무엇인가?

Server = 기능을 제공하는 것

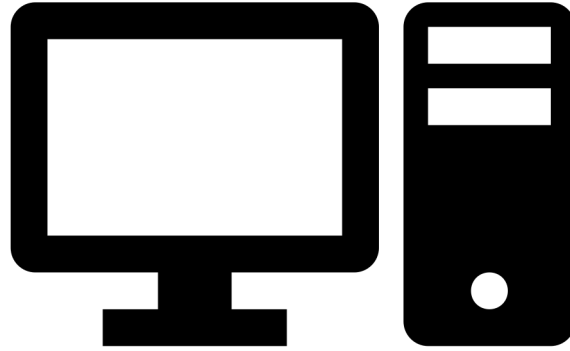
회원가입 기능 / 정보 가져오기 기능 / 추천 기능

# 서버란 무엇인가?

Server = 기능을 제공하는 것

회원가입 기능 / 정보 가져오기 기능 / 추천 기능

# 서버란 무엇인가?



회원가입 기능 / 정보 가져오기 기능 / 추천 기능



# 서버란 무엇인가?

사람 대신 컴퓨터가 이런 기능을 수행해준다



회원가입 기능 / 정보 가져오기 기능 / 추천 기능

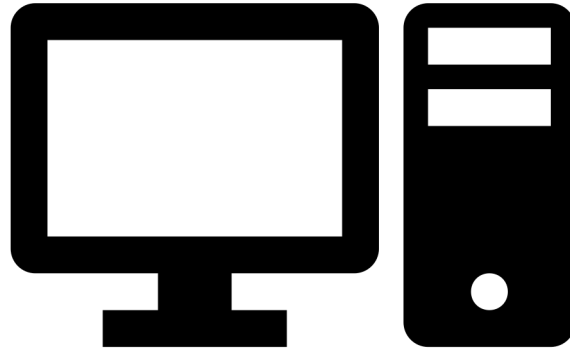
# 서버란 무엇인가?

사람 대신 **컴퓨터**가 이런 **기능을 수행**해준다



회원가입 기능 / 정보 가져오기 기능 / 추천 기능

# 서버란 무엇인가?



기능하는 컴퓨터 자체를 **서버**라고도 한다.

# 서버란 무엇인가? (정리)

어떠한 **기능을 제공**하는 프로그램

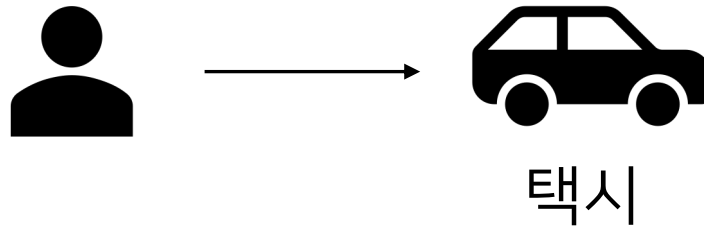
그 프로그램을 실행시키고 있는 **컴퓨터**

# 그런데 말입니다...

기능을 제공하기 위해서는 누군가의 **요청**이 있어야 한다.

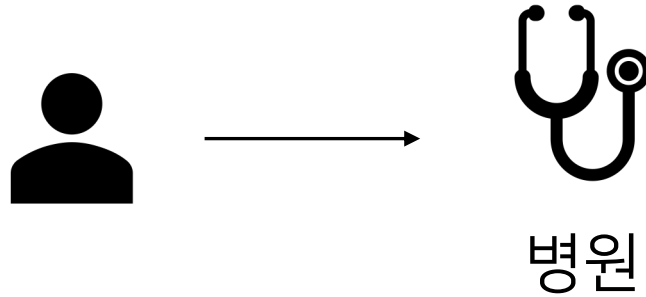
# 서버와 요청

기능을 제공하기 위해서는 누군가의 **요청**이 있어야 한다.



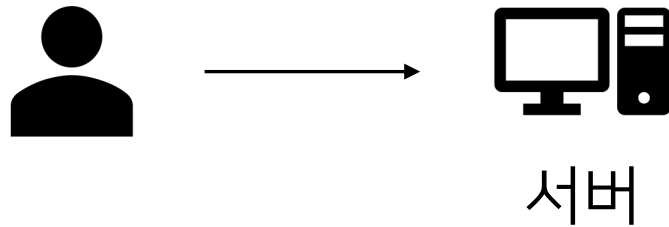
# 서버와 요청

기능을 제공하기 위해서는 누군가의 **요청**이 있어야 한다.



# 서버와 요청

서버에게도 **요청**을 해야 정해진 기능을 수행한다.





# 서버와 요청

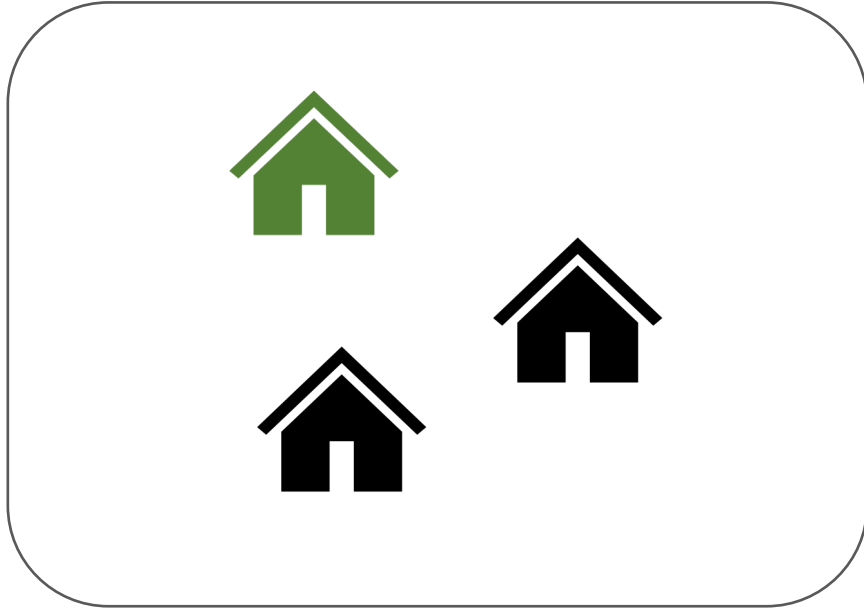
엇.. 그런데 컴퓨터에게 요청을 어떻게 하나?!

# 인터넷!

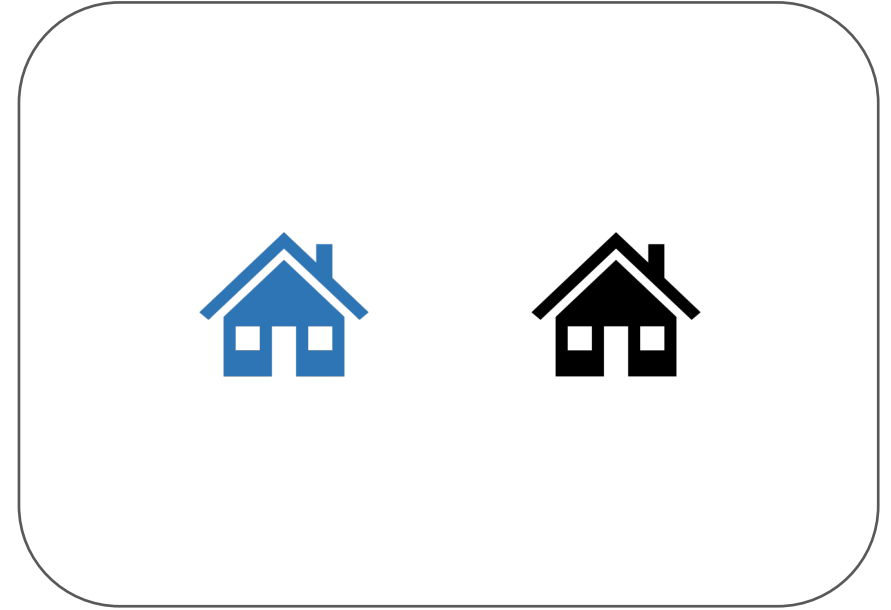


# **3강. 네트워크란 무엇인가?!**

# 이세계를 생각해봅시다!



A 부족

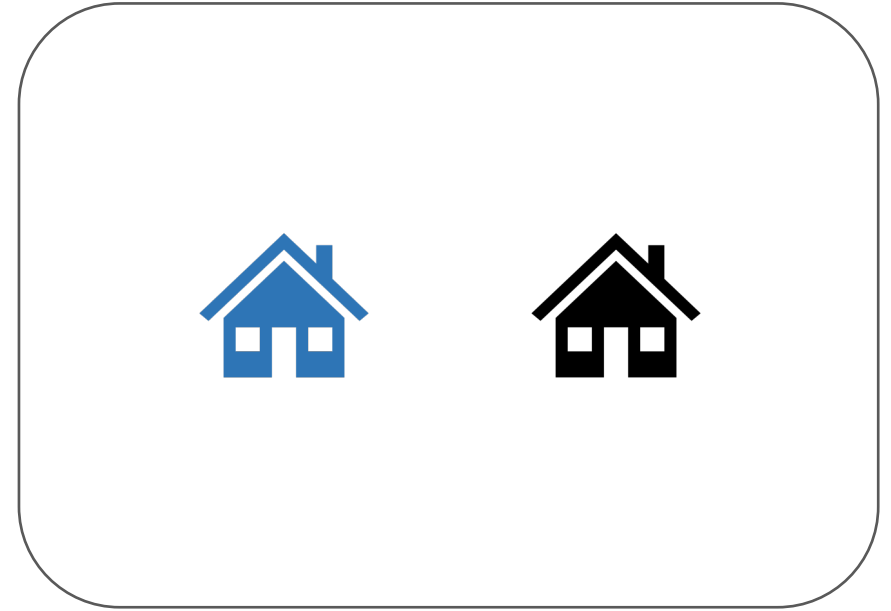


B 부족

이 이세계는 주소 체계가 잘 발달 되어 있다!

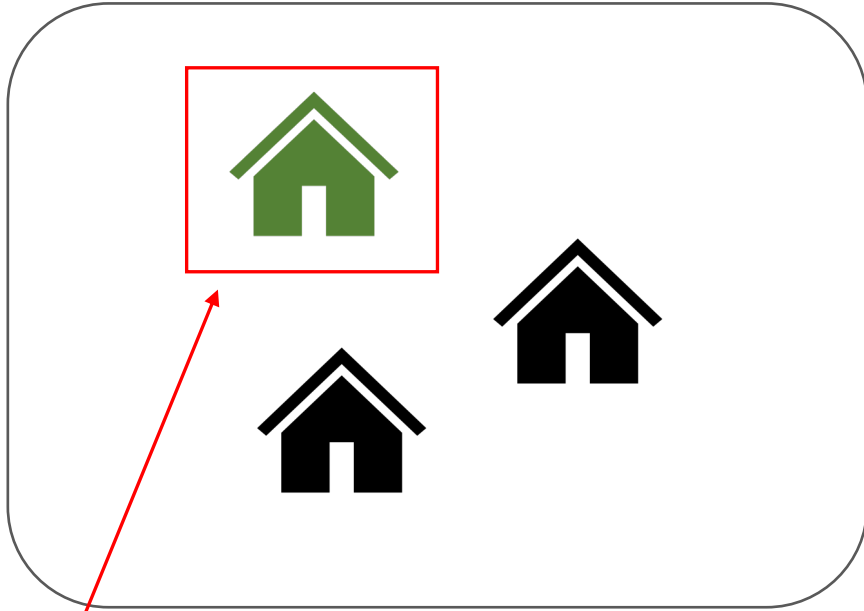


A 부족

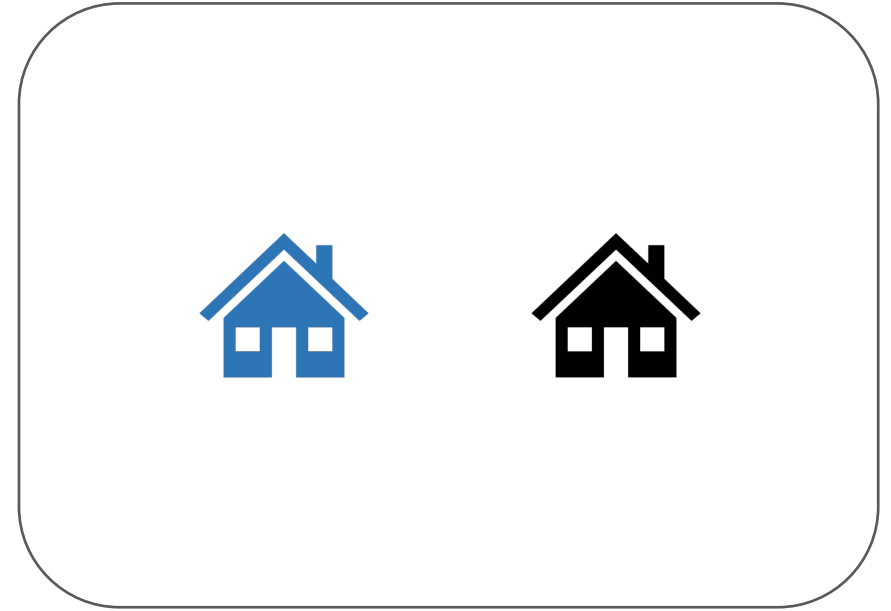


B 부족

# 이 이세계는 주소 체계가 잘 발달 되어 있다!



A 부족



B 부족

A부족 사과동 호랑이로 42번길 10

이 이세계는 주소 체계가 잘 발달 되어 있다!



A 부족



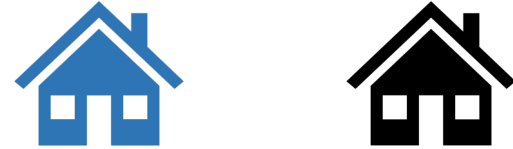
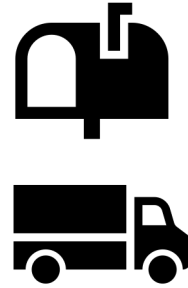
B 부족

B부족 감자동 곰로 13번길 2

# 심지어 이 이세계는 **택배 시스템**이 있다!!



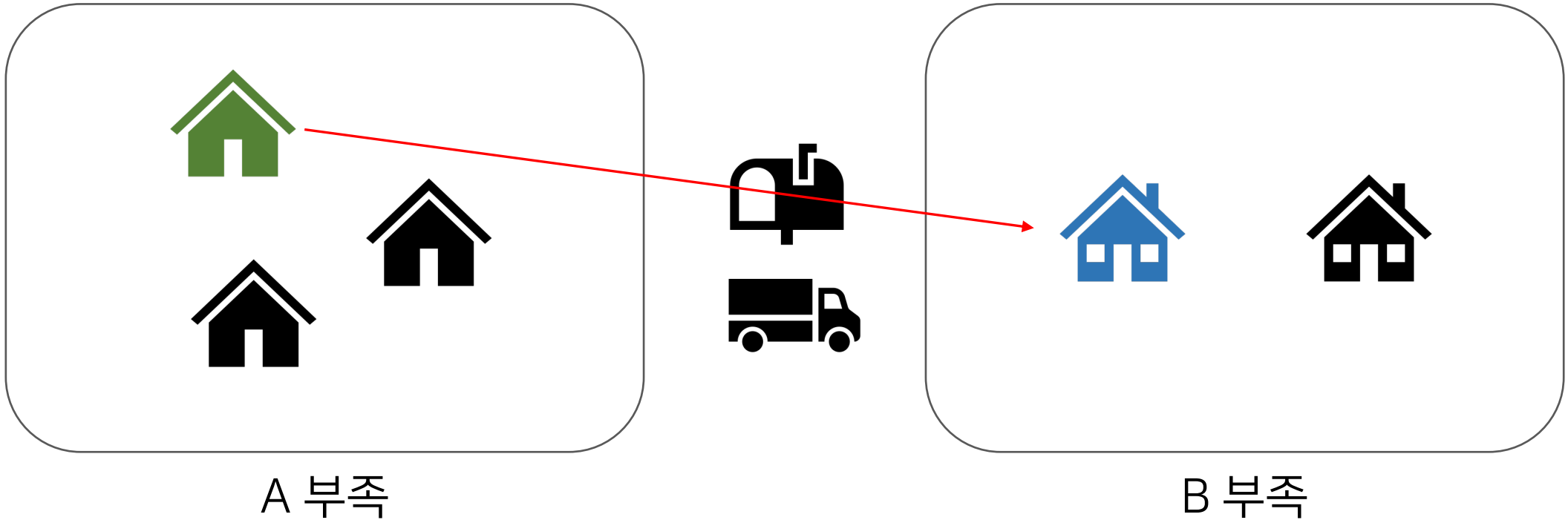
A 부족



B 부족



# 자 이제 A부족에서 B부족에 택배를 보내봅시다!



# 택배 받는 사람

이때 택배 받는 사람을 이렇게 쓸 거예요!

B부족 감자동 곰로 13번길 2에 사는 둘째

# 택배 받는 사람

이때 택배 받는 사람을 이렇게 쓸 거예요!

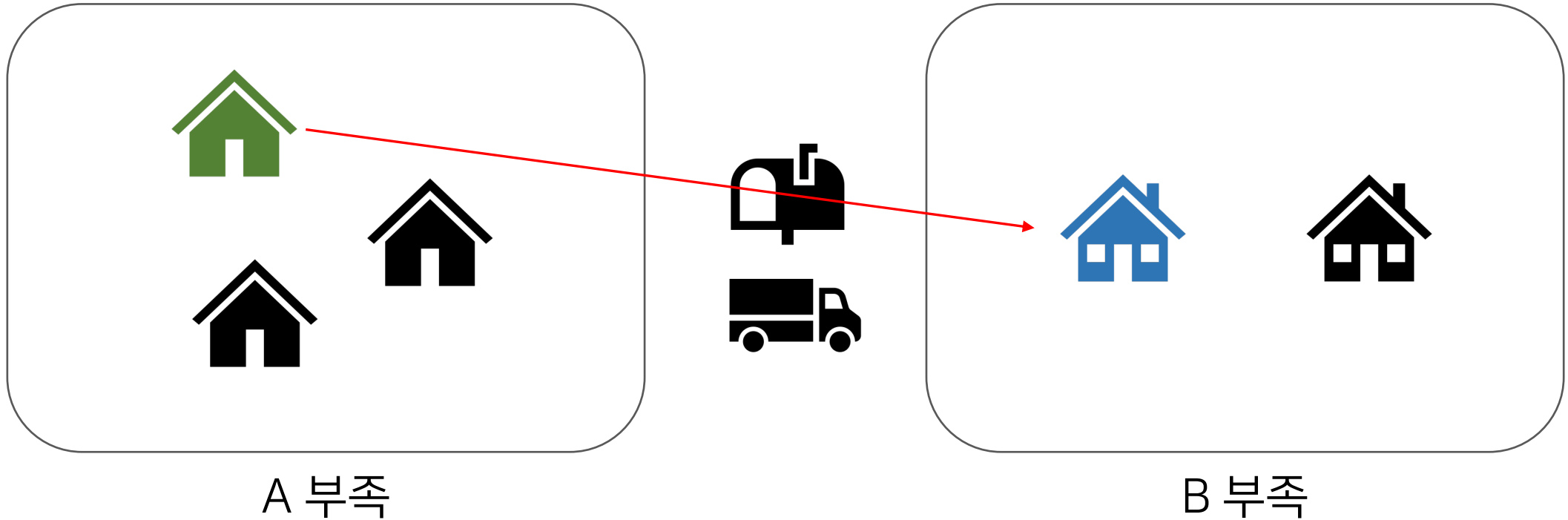
B부족 감자동 곰로 13번길 2에 사는 둘째

# 택배 받는 사람

이때 택배 받는 사람을 이렇게 쓸 거예요!

B부족 감자동 곰로 13번길 2에 사는 둘째

# 택배 받는 집을 잘 기억하는 방법이 있을까?



# 택배 받는 집을 잘 기억하는 방법이 있을까?

B부족 감자동 곰로 13번길 2 둘째는 너무 외우기 어려우니

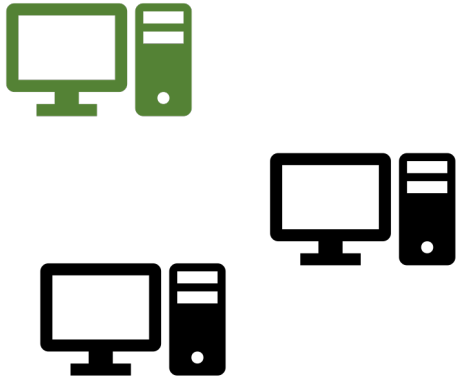
파란집 둘째 라고 축약!

# 택배 받는 집을 잘 기억하는 방법이 없을까?

B부족 감자동 곰로 13번길 2 둘째는 너무 외우기 어려우니

파란집 둘째 라고 축약!

# 이제 잠시 현실 세계로 넘어오겠습니다!



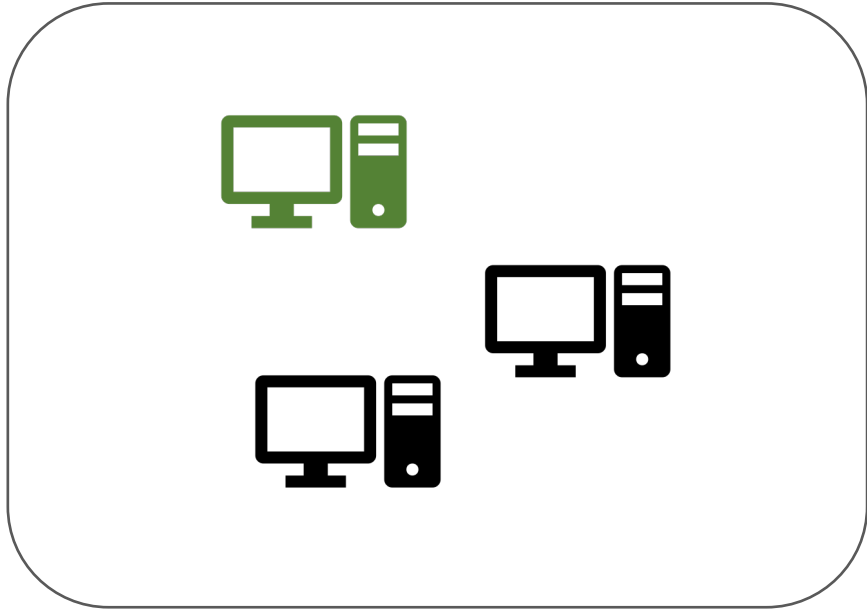
서울



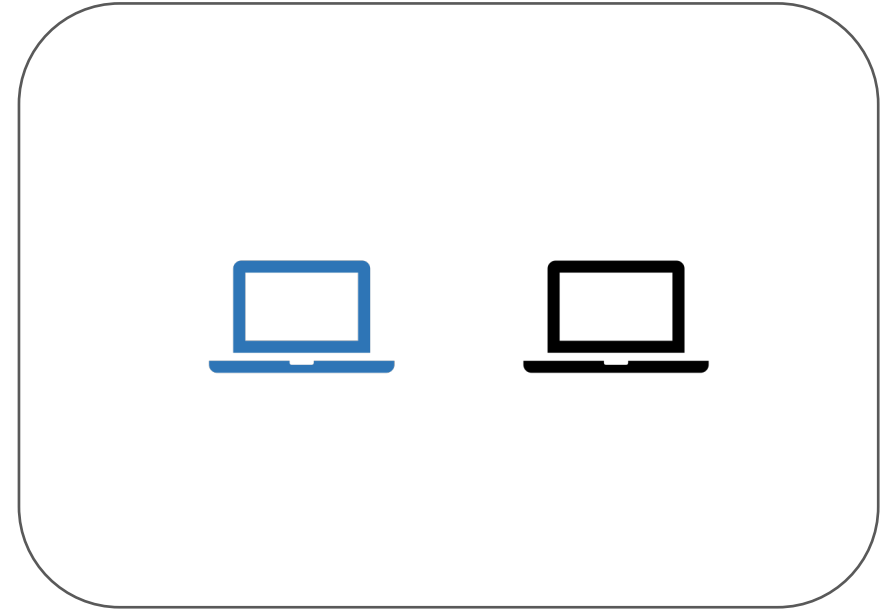
대전



우리 현실세계에서는 컴퓨터별 **고유 주소 (IP)**가 있습니다.

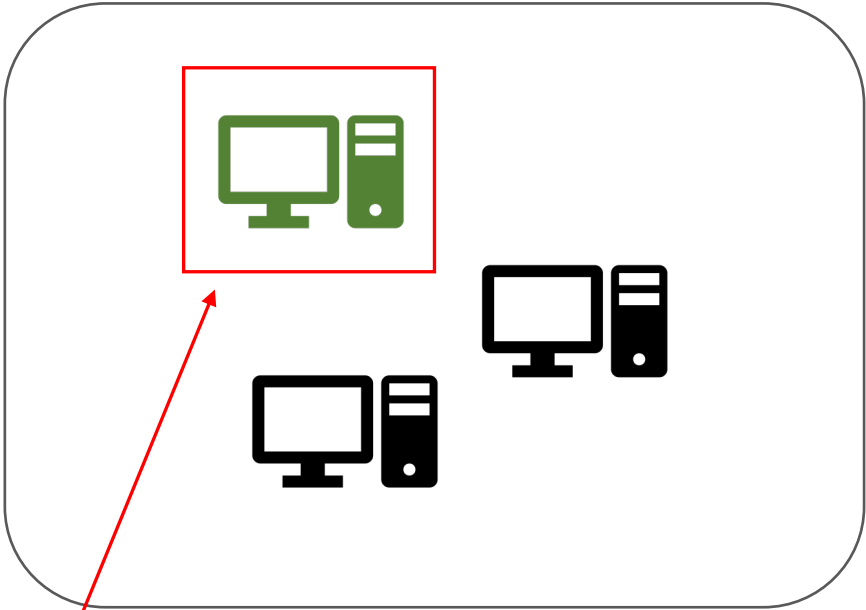


서울



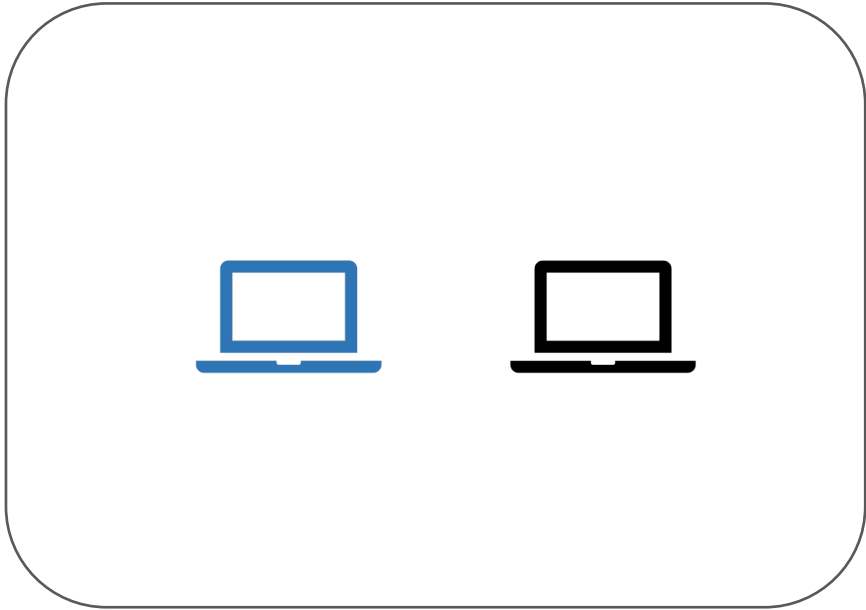
대전

우리 현실세계에서는 컴퓨터별 고유 주소 (IP)가 있습니다.



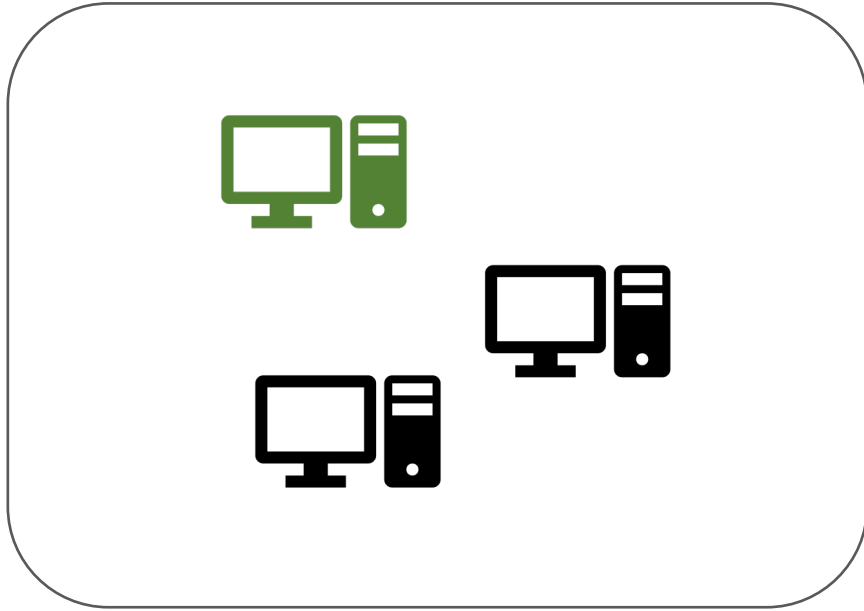
서울

123.1.22.19

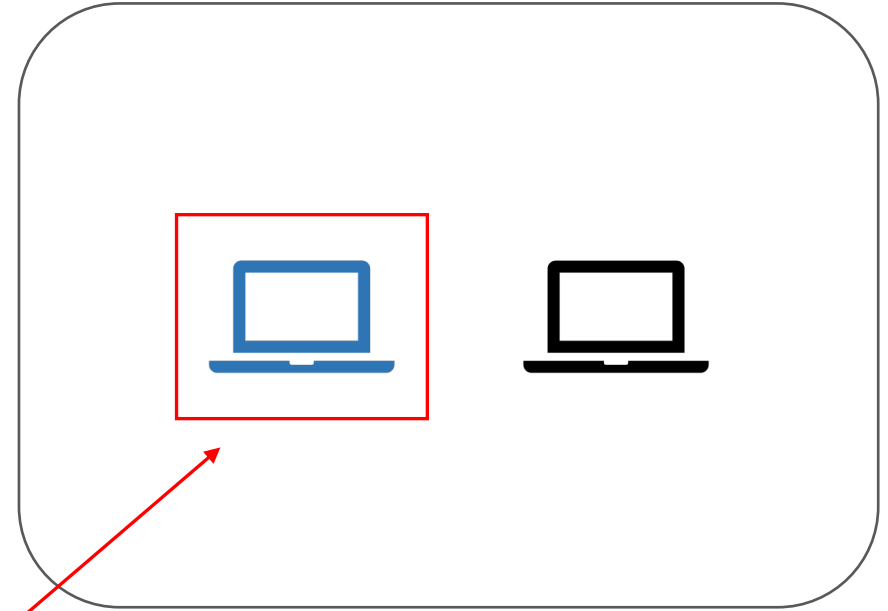


대전

우리 현실세계에서는 컴퓨터별 **고유 주소 (IP)**가 있습니다.

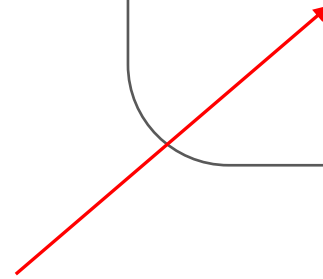


서울



대전

244.66.51.9



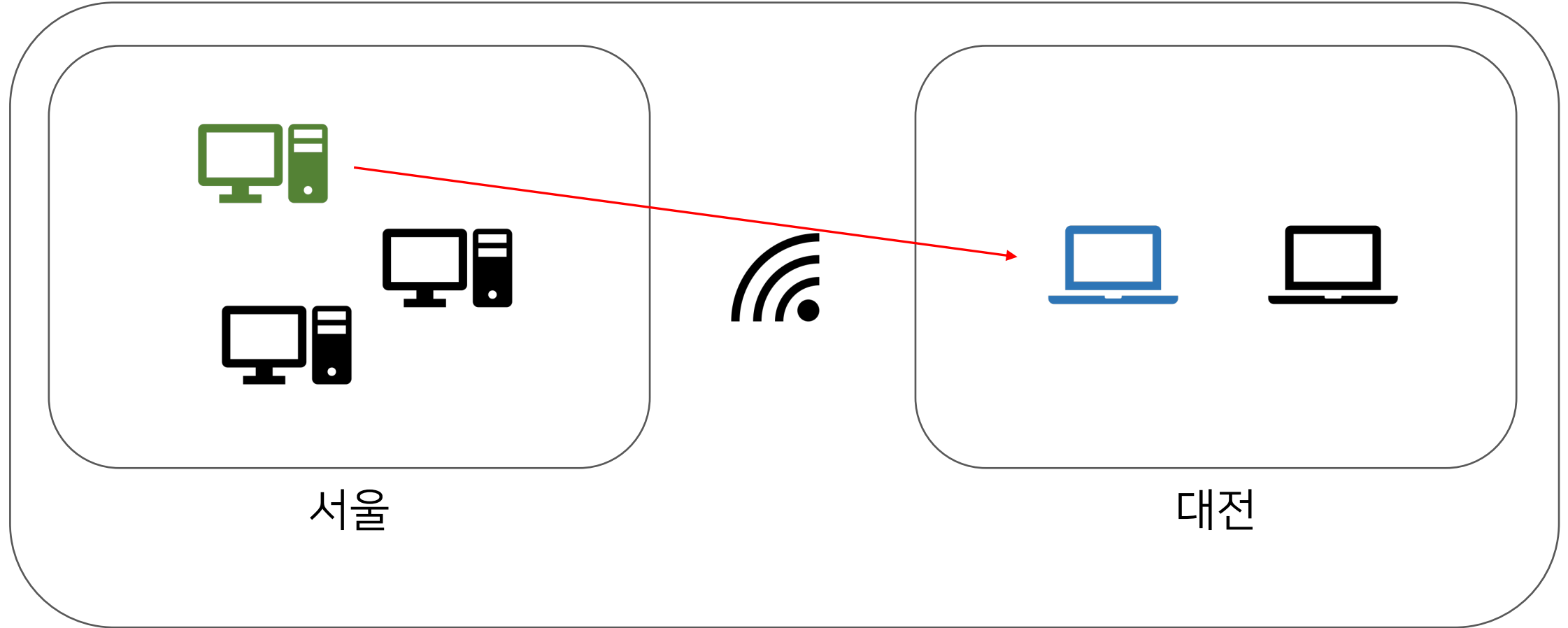
# 우리의 현실세계는 인터넷이 잘 발달되어 있습니다.



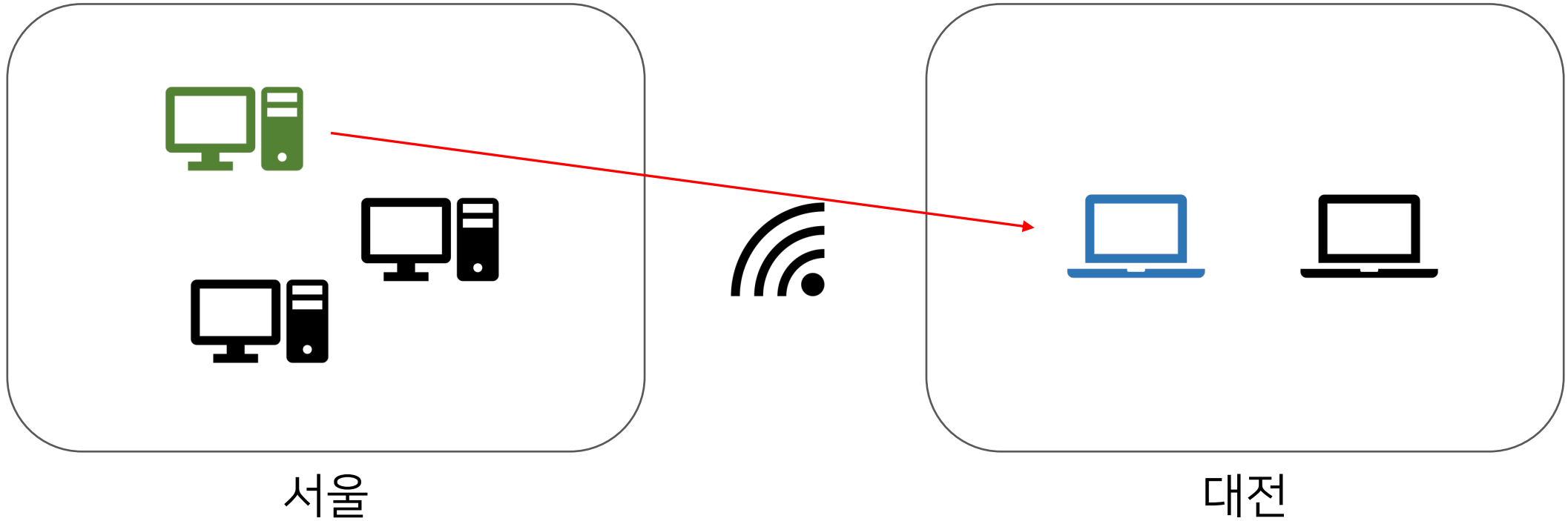
서울

대전

**또한 인터넷을 통해 데이터를 주고 받을 수 있습니다.**



# 서울에서 대전으로 데이터를 보내봅시다!



# 데이터 받는 컴퓨터

이번에 데이터 받는 컴퓨터는 이렇게 표현할 거예요!

IP 244.66.51.9, port : 3000

# 데이터 받는 컴퓨터

이번에 데이터 받는 컴퓨터는 이렇게 표현할 거예요!

IP 244.66.51.9, port: 3000

둘째처럼 3000번 포트를  
사용하는 프로그램이 데이터를 받는다.





# 데이터 받는 컴퓨터

이번에 데이터 받는 컴퓨터는 이렇게 표현할 거예요!

IP 244.66.51.9, port : 3000

# 244.66.51.9라는 숫자는 너무 외우기 어렵다!

IP 244.66.51.9, port : 3000

# Domain Name 등장!!

244.66.51.9 라는 외우기 어려운 숫자 대신,  
사람들이 외우기 쉬운 '이름'을 넣자!!

도메인이름 : spring.com, port : 3000

# Domain Name 등장!!

244.66.51.9 라는 외우기 어려운 숫자 대신,  
사람들이 외우기 쉬운 '이름'을 넣자!!

도메인이름 : spring.com, port : 3000

# Domain Name System (DNS)

IP 244.66.51.9

=

도메인 이름 spring.com

# 여기까지 중간 정리!

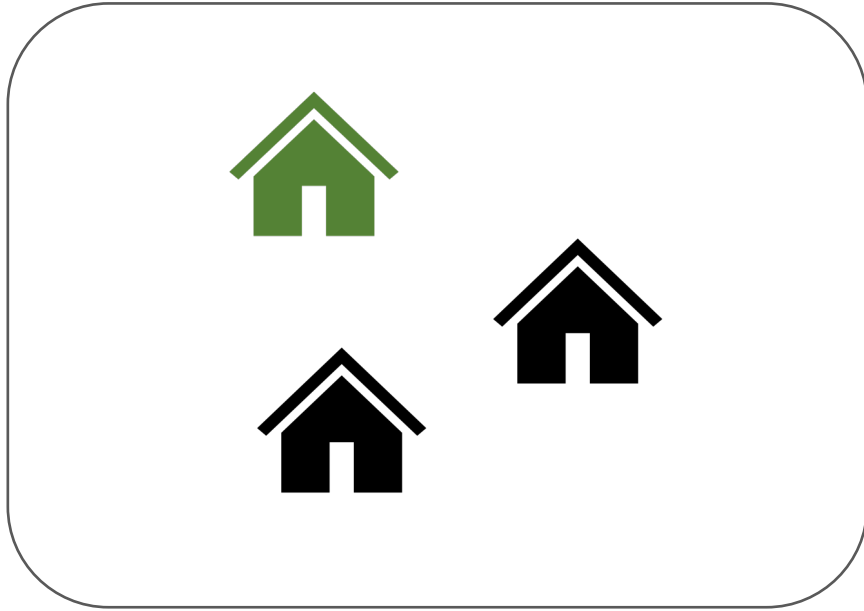
이세계 

현실 세계 

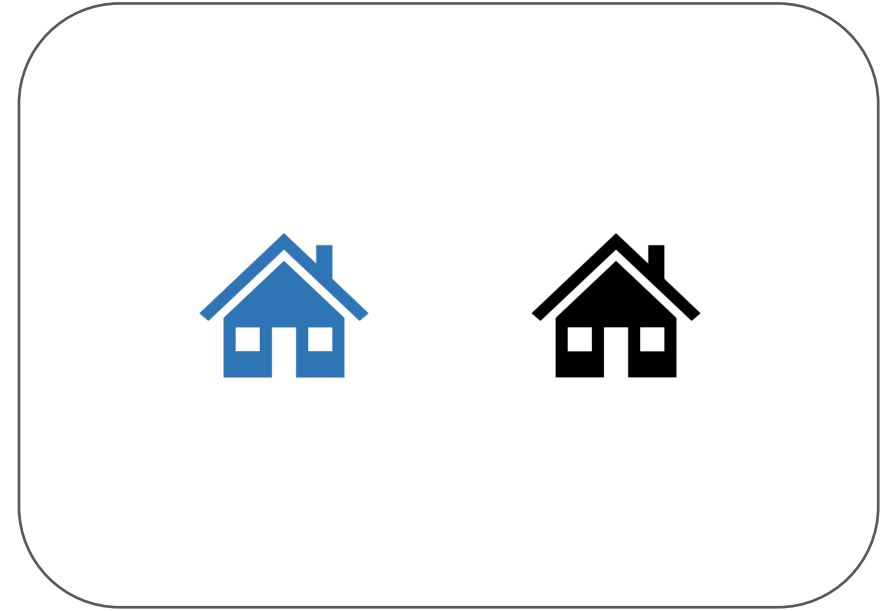
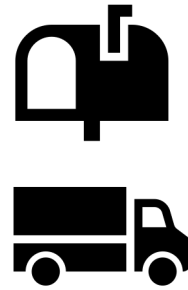
택배 시스템	네트워크
집	컴퓨터
주소 / B부족 감자동 곰로 13번길 2	IP / 244.77.51.9
집주소 별칭 / 파란집	도메인 이름 / spring.com
택배를 정말 받는 사람 / 둘째	port / 3000

# 4강. HTTP와 API란 무엇인가?!

# 택배(우체국) 시스템이 잘 잡혀있는 이세계를 보자!



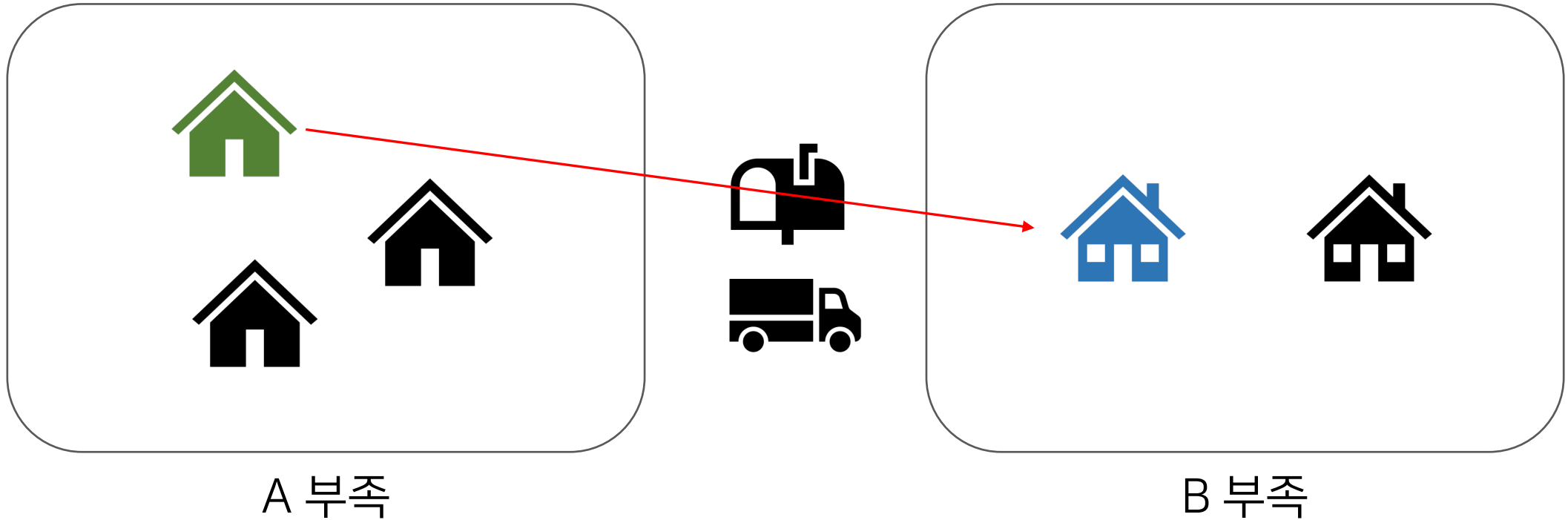
A 부족



B 부족



# 택배(편지)를 보내려면 무엇이 필요할까요?!

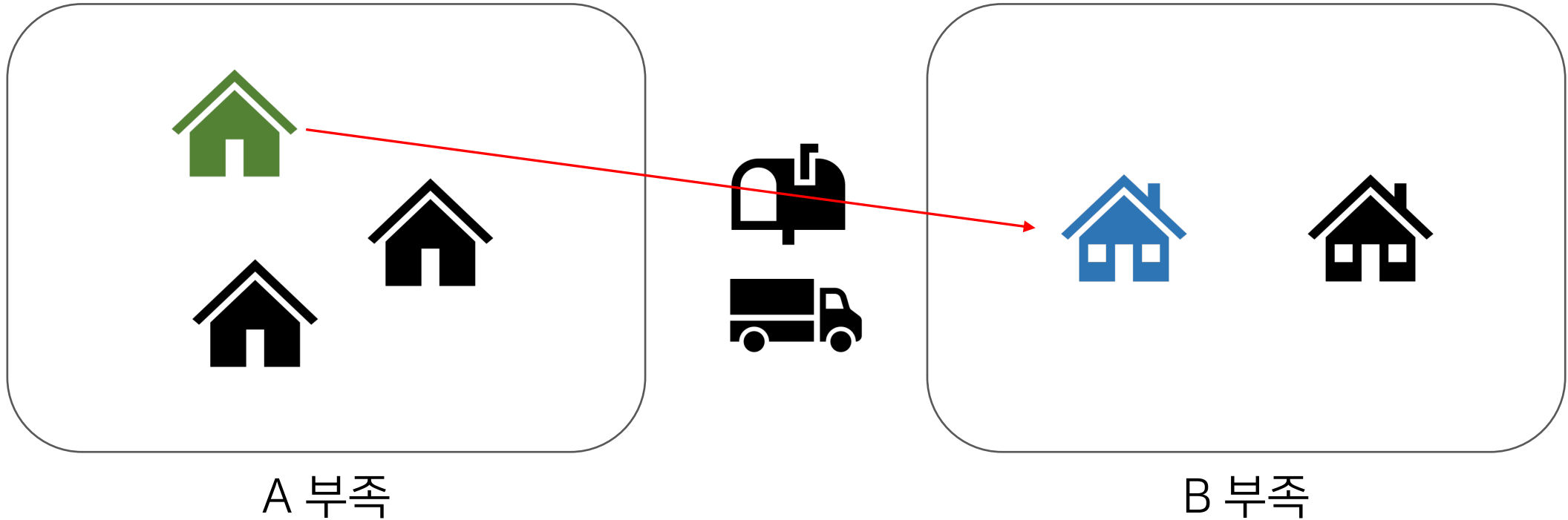


# 택배(편지)를 보내려면 무엇이 필요할까요?!

운송장

즉, 택배를 보내기 위한 **표준**이 있다!

**이세계의 운송장 표준은 다음과 같다.**



# 이세계의 운송장 표준은 다음과 같다.

운송장

내놓아라 파란집 둘째, 포션 빨강색 2개

# 이세계의 운송장 표준은 다음과 같다.

운송장

내놓아라 파란집 둘째, 포션 빨강색 2개

운송장을 받는 사람에게 요청하는 행위

# 이세계의 운송장 표준은 다음과 같다.

운송장

내놓아라 파란집 둘째, 포션 빨강색 2개

운송장이 가는 집

# 이세계의 운송장 표준은 다음과 같다.

운송장

내놓아라 파란집 둘째, 포션 빨강색 2개

운송장을 실제 받는 사람

# 이세계의 운송장 표준은 다음과 같다.

운송장

내놓아라 파란집 둘째, 포션 빨강색 2개

운송장을 받는 사람에게 원하는 자원



# 이세계의 운송장 표준은 다음과 같다.

운송장

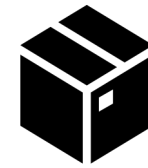
내놓아라 파란집 둘째, 포션 빨강색 2개

자원의 세부 조건

# 다른 표준도 하나 살펴보자!

운송장

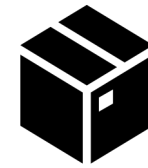
창고에 넣어라 빨간집, 오크가죽



# 다른 표준도 하나 살펴보자!

운송장

창고에 넣어라 빨간집, 오크가죽

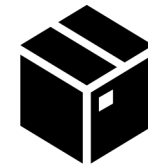


운송장을 받는 사람에게 요청하는 행위

# 다른 표준도 하나 살펴보자!

운송장

창고에 넣어라 빨간집, 오크가죽

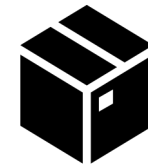


운송장을 받는 사람 (한 명만 살고 있다!)

# 다른 표준도 하나 살펴보자!

운송장

창고에 넣어라 빨간집, 오크가죽

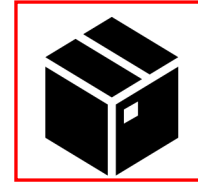


운송장을 받는 사람에게 원하는 **자원**

# 다른 표준도 하나 살펴보자!

운송장

창고에 넣어라 빨간집, 오크가족

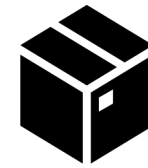


실제 오크 가족이 들어 있는 박스

# 다른 표준도 하나 살펴보자!

운송장

창고에넣어라 빨간집, 오크가죽



행위와 자원은 빨간집에 운송장을 보내기 전에 약속해야 한다!!

# 다시 현실세계로 돌아 오겠습니다!

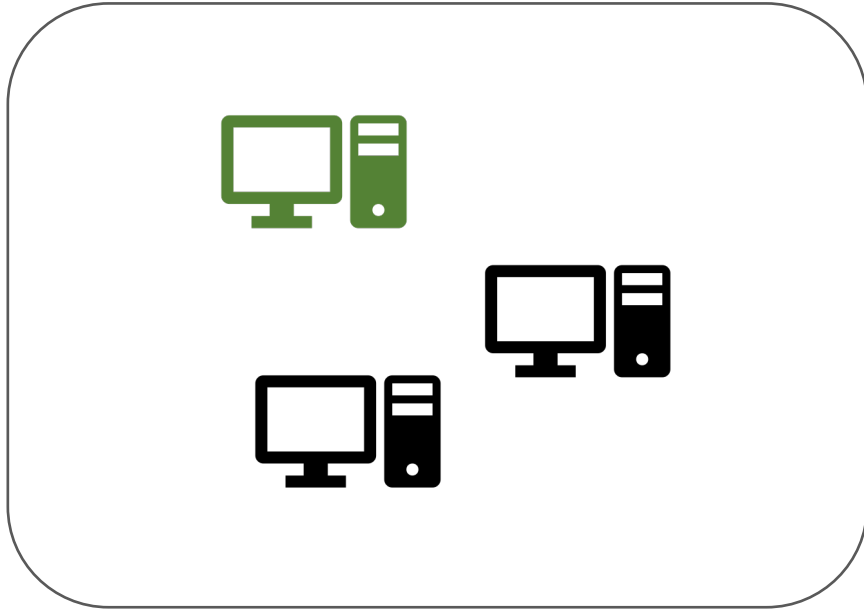


서울

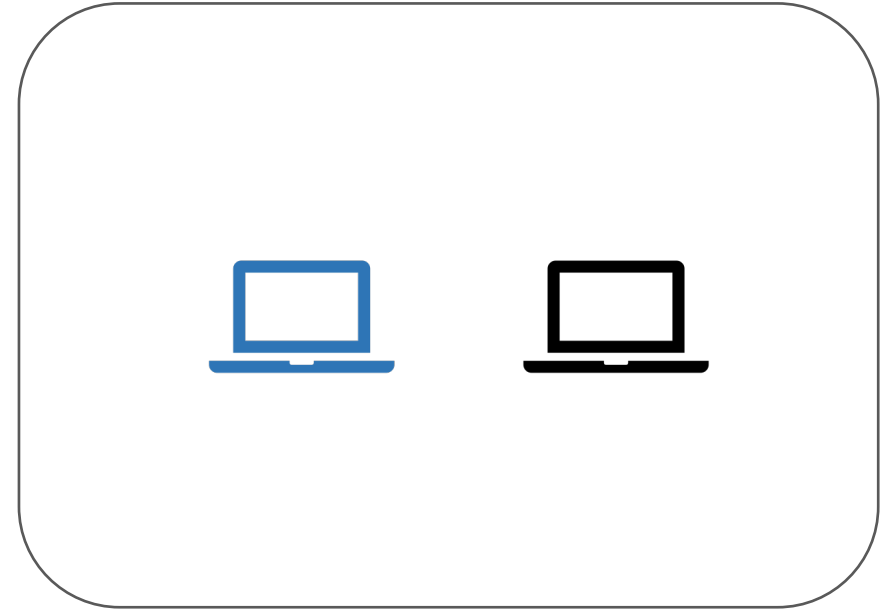
대전



# 현실세계에서도 데이터를 주고 받는 표준이 있습니다!



서울



대전

**현실세계에서도 데이터를 주고 받는 표준이 있습니다!**

**HTTP** (HyperText Transfer Protocol)

# 현실세계에서도 데이터를 주고 받는 표준이 있습니다!

HTTP (HyperText Transfer Protocol)

Protocol : 표준, 약속

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2  
Host: spring.com:3000
```

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2  
Host: spring.com:3000
```

HTTP 요청을 받는 컴퓨터에게 요청하는 **행위** (데이터를 달라)

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2  
Host: spring.com:3000
```

HTTP Method 라고 부릅니다!

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2
```

```
Host: spring.com:3000
```

HTTP 요청을 받는 컴퓨터와 프로그램 정보

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2  
Host: spring.com:3000
```

HTTP 요청을 받는 컴퓨터에게 원하는 **자원**



# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2  
Host: spring.com:3000
```

Path라고 부릅니다!

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2  
Host: spring.com:3000
```

구분기호

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2  
Host: spring.com:3000
```

자원의 세부 조건 (색깔은 빨간색)

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

```
GET /portion?color=red&count=2  
Host: spring.com:3000
```

구분기호

# 이 표준 역시 지켜야할 규칙이 있다.



HTTP 요청

GET /portion?color=red&count=2

Host: spring.com:3000

자원의 세부 조건 (개수는 2개)

# 또 다른 예시를 들어보죠~!



HTTP 요청

POST /oak/leather

Host: spring.com:3000

오크가족정보

# 또 다른 예시를 들어보죠~!



HTTP 요청

**POST** /oak/leather

Host: spring.com:3000

오크가족정보

HTTP 요청을 받는 컴퓨터에게 요청하는 **행위** (저장하라)

# 또 다른 예시를 들어보죠~!



HTTP 요청

POST /oak/leather

Host: spring.com:3000

오크가족정보

HTTP 요청을 받는 컴퓨터와 프로그램 정보



# 또 다른 예시를 들어보죠~!



HTTP 요청

POST /oak/leather

Host: spring.com:3000

오크가족정보

HTTP 요청을 받는 컴퓨터에게 원하는 **자원**

# 또 다른 예시를 들어보죠~!



HTTP 요청

POST /oak/leather

Host: spring.com:3000

오크가족정보

실제 저장할 오크 가족 정보 (문법은 다음 시간에!)

# 또 다른 예시를 들어보죠~!



HTTP 요청

POST /oak/leather

Host: spring.com:3000

오크가족정보

행위와 자원은 HTTP 요청을 보내기 전에 약속해야 한다!!

# 자 뭐가 비슷하죠?! 중간 정리 한 번 해보겠습니다!

이세계 

현실 세계 

운송장

HTTP

내놓아라 파란집 둘째, 포션 빨강색 2개

GET /portion?color=red&count=2  
Host: spring.com:3000

# 자 뭐가 비슷하죠?! 중간 정리 한 번 해보겠습니다!

이세계 

현실 세계 

운송장


HTTP

내놓아라 파란집 둘째, 포션 빨강색 2개

GET /portion?color=red&count=2  
Host: spring.com:3000

Http Method와 Path

# 자 뭐가 비슷하죠?! 중간 정리 한 번 해보겠습니다!

이세계 

현실 세계 

운송장

HTTP

내놓아라 파란집 둘째, 포션 빨강색 2개

GET /portion?color=red&count=2  
Host: spring.com:3000

원하는 조건 (Query라고 부릅니다!)

# 자 뭐가 비슷하죠?! 중간 정리 한 번 해보겠습니다!

이세계 

현실 세계 

운송장


HTTP

내놓아라 파란집 둘째, 포션 빨강색 2개

GET /portion?color=red&count=2  
Host: spring.com:3000

HTTP 요청을 받는 컴퓨터와 프로그램

# 자 뭐가 비슷하죠?! 중간 정리 한 번 해보겠습니다!

이세계 

현실 세계 

운송장

HTTP

창고에넣어라 빨간집, 오크가죽



POST /oak/leather  
Host: spring.com:3000

오크가죽정보



# 자 뭐가 비슷하죠?! 중간 정리 한 번 해보겠습니다!

이세계 

현실 세계 

운송장

HTTP

창고에 넣어라 빨간집, 오크가족



POST /oak/leather  
Host: spring.com:3000

오크가족정보

바디(**Body**)라고 부른다.

**추가적으로 몇 가지 개념을 다루어보겠습니다!**

# 쿼리와 바디, 정보를 보내는 2가지 방법!

## 쿼리

GET /portion?color=red&count=2  
Host: spring.com:3000

## 바디

POST /oak/leather  
Host: spring.com:3000

오크가죽정보

# 쿼리와 바디, 정보를 보내는 2가지 방법!

## 쿼리

GET /portion?color=red&count=2  
Host: spring.com:3000

## 바디

POST /oak/leather  
Host: spring.com:3000

오크가족정보

# 다양한 HTTP Method

GET : 데이터를 달라, 쿼리  
POST : 데이터를 저장하라, 바디

PUT : 데이터를 수정하라, 바디  
DELETE : 데이터를 삭제하라, 쿼리

# API (Application Programming Interface)



GET /portion?color=red&count=2  
Host: spring.com:3000

POST /oak/leather  
Host: spring.com:3000

오크가죽정보

정해진 약속을 하여, 특정 기능을 수행하는 것

# HTTP 요청 문법



첫째줄 (메소드 패스 쿼리)  
+ HTTP 버전

```
POST /oak/leather  
Host: spring.com:3000
```

오크가족정보

# HTTP 요청 문법



첫째줄 (메소드 패스 쿼리)

POST /oak/leather
Host: spring.com:3000

헤더 (여러줄 가능)

오브젝트 정보



# HTTP 요청 문법



첫째줄 (메소드 패스 쿼리)

POST /oak/leather

Host: spring.com:3000

한 줄 띄기

오크가족정보

헤더 (여러줄 가능)

# HTTP 요청 문법



첫째줄 (메소드 패스 쿼리)

POST /oak/leather

Host: spring.com:3000

한 줄 띄기

오크가족정보

헤더 (여러줄 가능)

바디 (여러줄 가능)

# URL

URL (Uniform Resource Locator)

<http://spring.com:3000/portion?color=red&count=2>

# URL

http://spring.com:3000/portion?color=red&count=2

사용하고 있는 프로토콜 (HTTP)

# URL

<http://spring.com:3000/portion?color=red&count=2>

구분기호

# URL

<http://spring.com:3000/portion?color=red&count=2>

도메인 이름:포트, 도메인 이름은 IP로 대체 가능하다!

# URL

<http://spring.com:3000/portion?color=red&count=2>

자원의 경로 (Path)

# URL

<http://spring.com:3000/portion?color=red&count=2>

구분기호



# URL

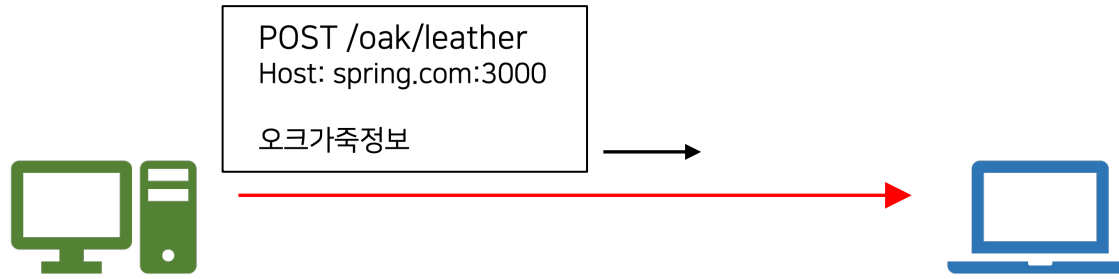
<http://spring.com:3000/portion?color=red&count=2>

color=red&count=2 쿼리 (추가정보)

# 요청을 받았으면 **응답**을 줘야죠?! **HTTP** **응답**



# 요청을 받았으면 응답을 줘야죠?! HTTP 응답



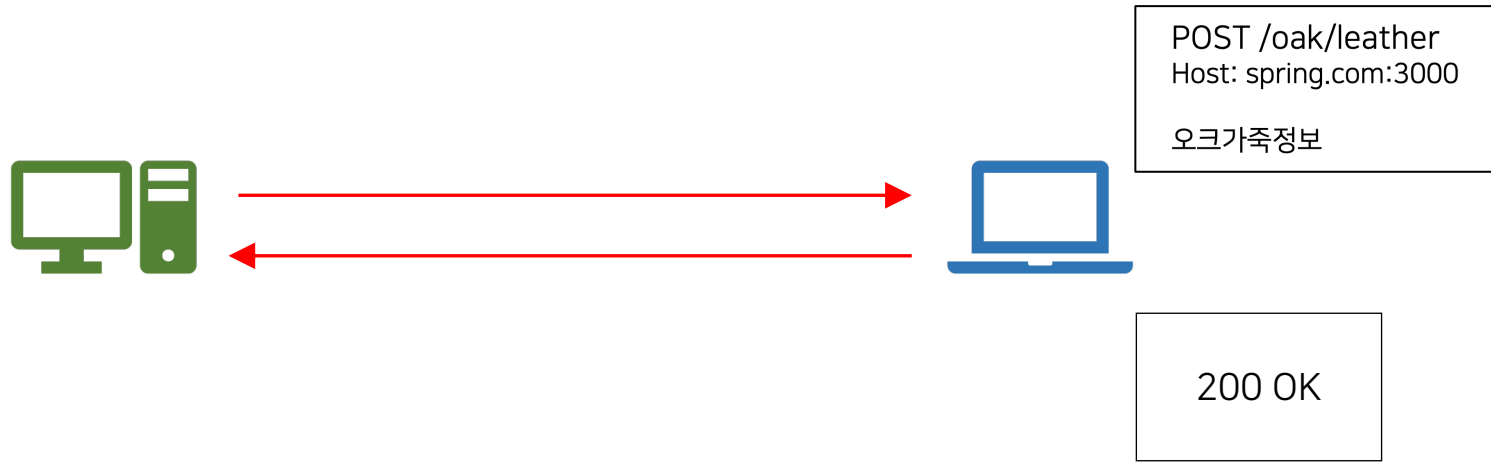
# 요청을 받았으면 응답을 줘야죠?! HTTP 응답



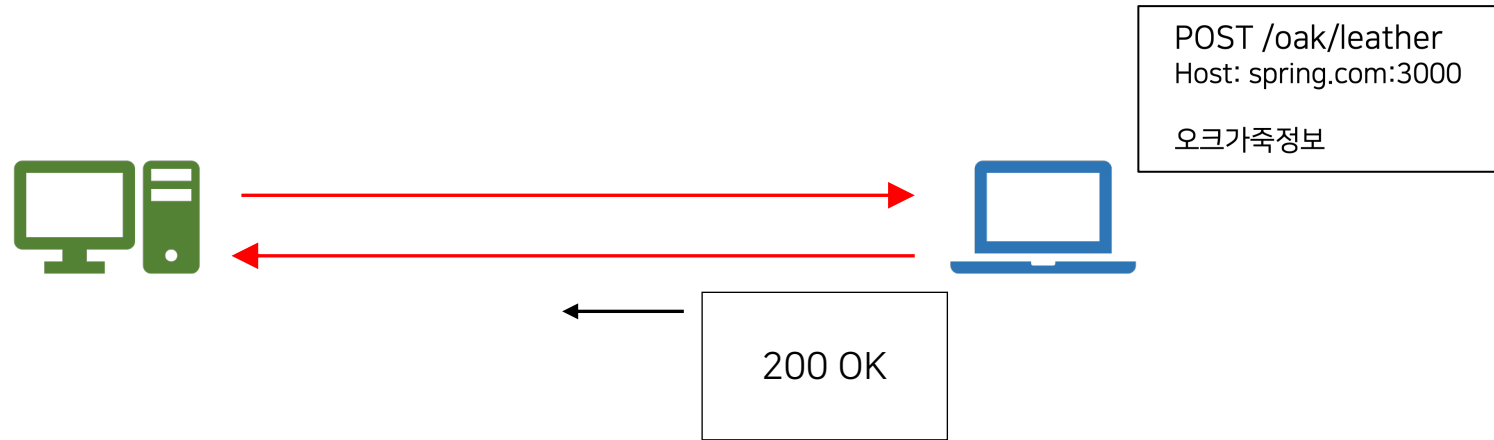
POST /oak/leather  
Host: spring.com:3000

오크가족정보

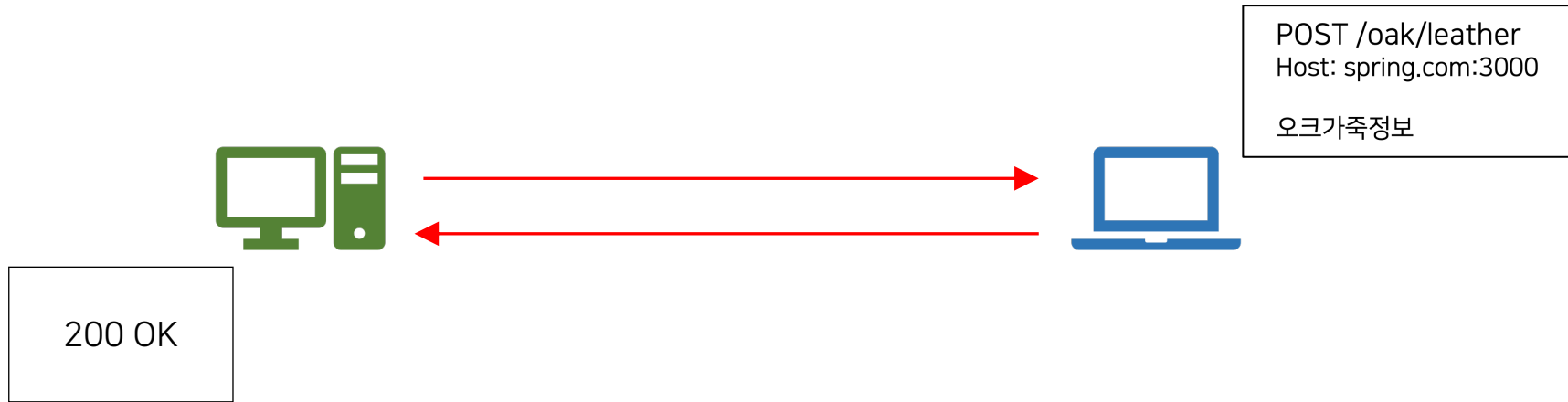
# 요청을 받았으면 응답을 줘야죠?! HTTP 응답

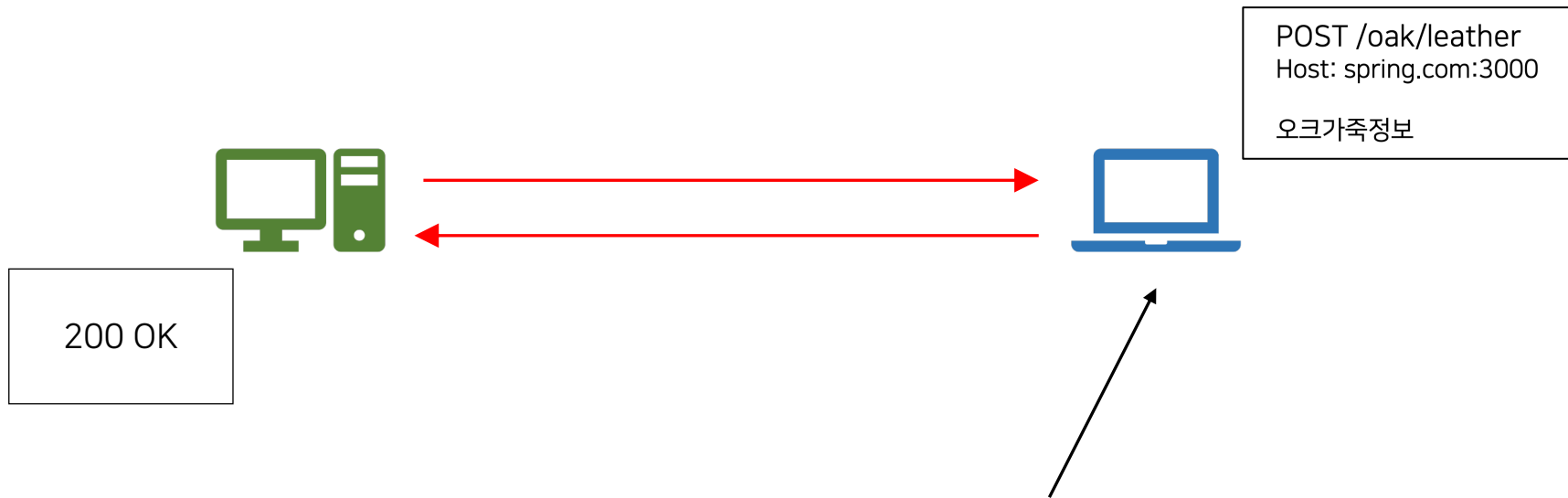


# 요청을 받았으면 응답을 줘야죠?! HTTP 응답



# 요청을 받았으면 응답을 줘야죠?! HTTP 응답

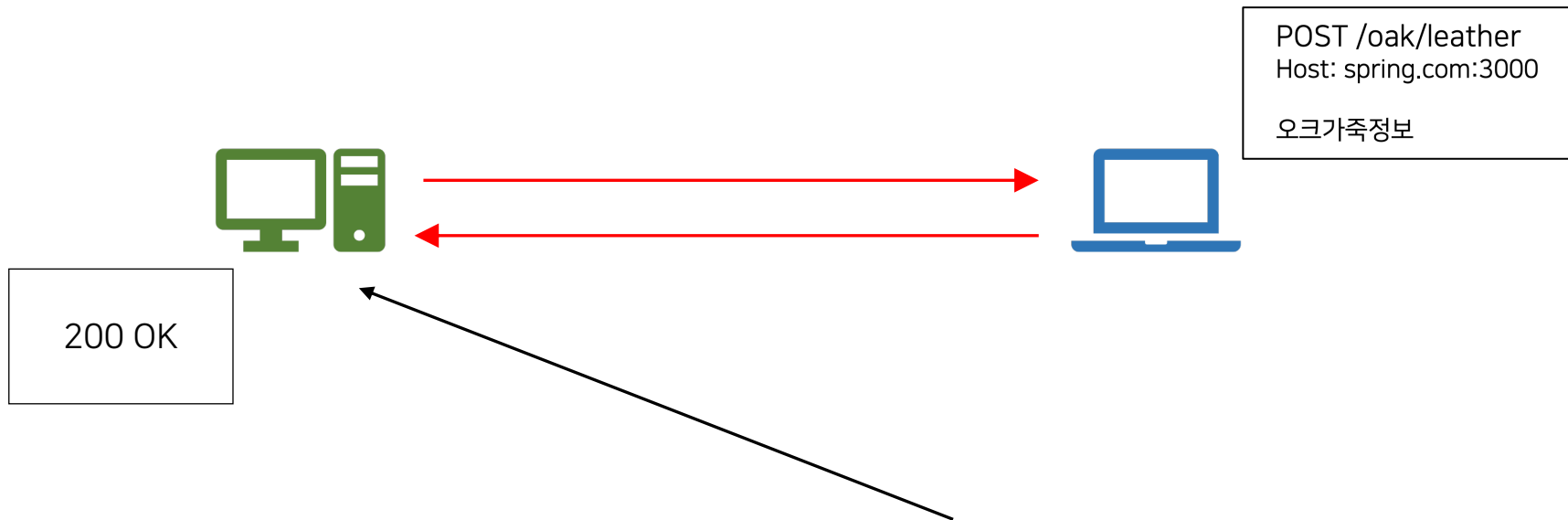




요청에 대한 응답을 제공(serve)한 컴퓨터! 바로 **서버**입니다!

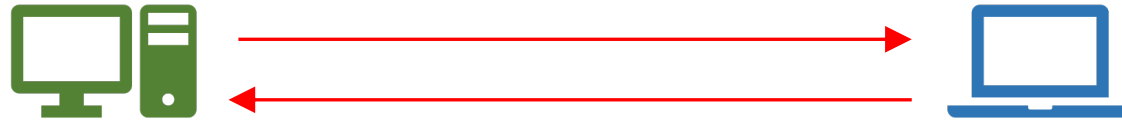


# 클라이언트 - 서버 구조



요청을 한 컴퓨터! 바로 고객님(**Client**) 컴퓨터입니다!

# 응답에 들어가는 숫자(상태 코드)는 매우 다양합니다!



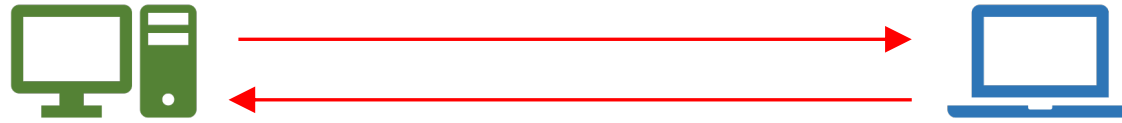
200 OK

300  
Moved Permanently

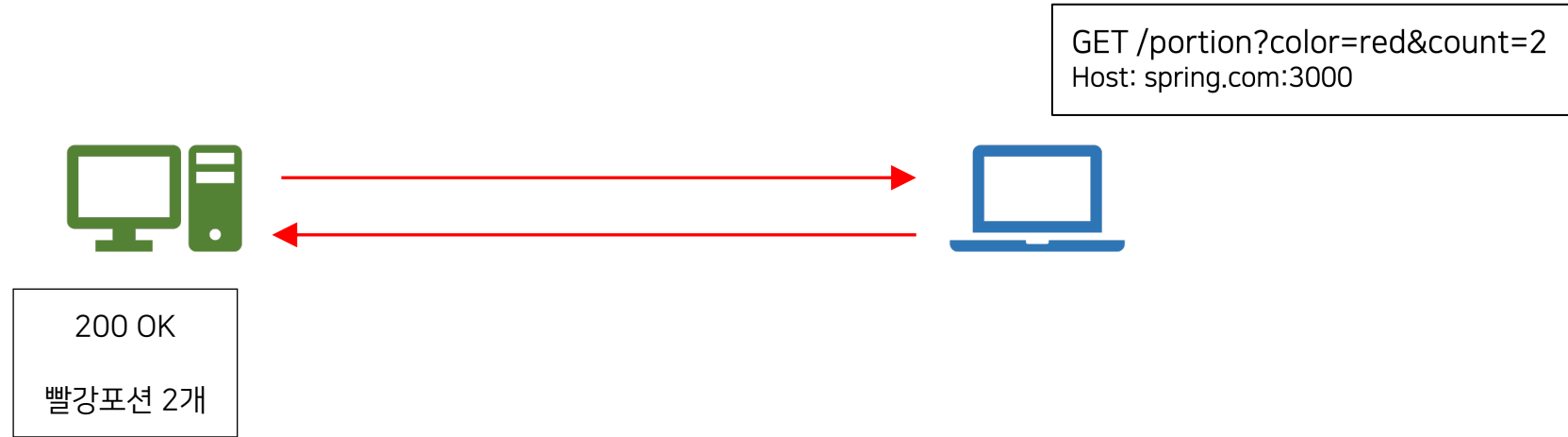
404  
NotFound

500  
Internal Server Error

응답에는 추가 정보(**바디**)를 담을 수도 있습니다.



# 응답에는 추가 정보(바디)를 담을 수도 있습니다.



# HTTP 응답 역시 요청과 구조가 동일합니다.

첫째줄 - 상태코드
여러줄 - 헤더
한 줄 띄기
여러줄 - 바디

# HTTP 응답 역시 요청과 구조가 동일합니다.

첫째줄 - 상태코드
여러줄 - 헤더
한 줄 띄기
여러줄 - 바디

```
HTTP/1.1 200 OK
Content-Type: application/json

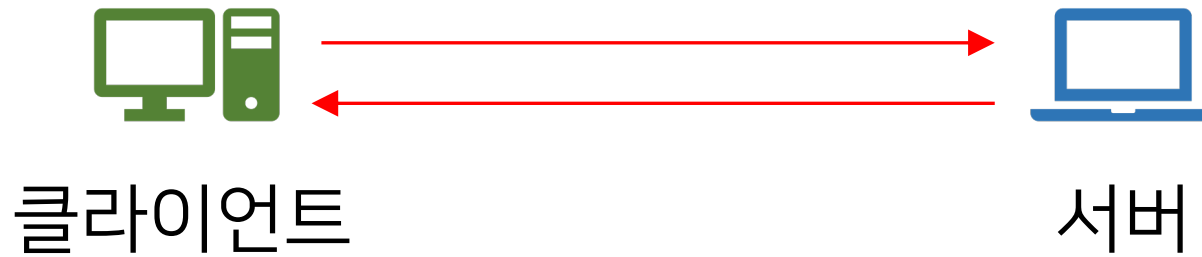
{
  "name": "A",
  "age": null
}
```

# 총정리

1. (웹을 통한) 컴퓨터 간의 통신은 HTTP라는 표준화된 방식이 있다.
2. HTTP 요청은 HTTP Method (GET, POST)와 Path (/portion)가 핵심이다.
3. 요청에서 데이터를 전달하기 위한 2가지 방법은 쿼리와 바디이다.
4. HTTP 응답은 **상태 코드**가 핵심이다.

# 총정리

5. 클라이언트와 서버는 HTTP를 주고 받으며 기능을 동작하는데 이때 정해진 규칙을 API라고 한다.





# 5강. GET API 개발하고 테스트하기

# API를 이루고 있는 요소

GET /portion?color=red&count=2  
Host: spring.com:3000

HTTP Method

# API를 이루고 있는 요소

GET /portion?color=red&count=2  
Host: spring.com:3000

HTTP Method

HTTP Path

# API를 이루고 있는 요소

GET /portion?color=red&count=2  
Host: spring.com:3000

HTTP Method

HTTP Path

쿼리 (key와 value)

# API를 이루고 있는 요소

GET /portion?color=red&count=2  
Host: spring.com:3000

HTTP Method

HTTP Path

쿼리 (key와 value)

API의 반환 결과

# API를 이루고 있는 요소

GET /portion?color=red&count=2  
Host: spring.com:3000

HTTP Method

HTTP Path

쿼리 (key와 value)

API의 반환 결과

# **API를 개발하기 전에 해야 할 것!**

앞서 살펴본 것들을 정해야 한다!

# API를 개발하기 전에 해야 할 것!

앞서 살펴본 것들을 정해야 한다!



API Specification (명세)



# API를 개발하기 전에 해야 할 것!

앞서 살펴본 것들을 정해야 한다!

API Spec



# 딱셈 API

HTTP Method

HTTP Path

쿼리 (key와 value)

API의 반환 결과

# 덧셈 API

HTTP Method  GET

HTTP Path

쿼리 (key와 value)

API의 반환 결과

# 덧셈 API

HTTP Method GET

HTTP Path  /add

쿼리 (key와 value)

API의 반환 결과

# 덧셈 API

HTTP Method

GET

HTTP Path

/add

쿼리 (key와 value) 

int number1 / int number2

API의 반환 결과

# 덧셈 API

HTTP Method

GET

HTTP Path

/add

쿼리 (key와 value)

int number1 / int number2

API의 반환 결과



숫자 - 두 숫자의 덧셈 결과

# 이제 실제 코딩을 해보자!

```
@RestController
public class CalculatorController {

    @GetMapping("/add")
    public int addTwoNumbers(
        @RequestParam int number1,
        @RequestParam int number2
    ) {
        return number1 + number2;
    }
}
```

# 이제 실제 코딩을 해보자!

```
@RestController
```

```
public class CalculatorController {
```

```
    @GetMapping("/add")
```

```
    public int addTwoNumbers(
```

```
        @RequestParam int number1,
```

```
        @RequestParam int number2
```

```
    ) {
```

```
        return number1 + number2;
```

```
    }
```

```
}
```

@RestController :

주어진 Class를 Controller로 등록한다.



# 이제 실제 코딩을 해보자!

```
@RestController
```

```
public class CalculatorController {
```

```
    @GetMapping("/add")
```

```
    public int addTwoNumbers(
```

```
        @RequestParam int number1,
```

```
        @RequestParam int number2
```

```
    ) {
```

```
        return number1 + number2;
```

```
    }
```

```
}
```

@RestController :

주어진 Class를 Controller로 등록한다.

Controller : **API의 입구**

# 이제 실제 코딩을 해보자!

```
@RestController
public class CalculatorController {

    @GetMapping("/add")
    public int addTwoNumbers(
        @RequestParam int number1,
        @RequestParam int number2
    ) {
        return number1 + number2;
    }
}
```

@GetMapping("/add") :

아래 함수를  
HTTP Method가 GET이고  
HTTP path가 /add인  
API로 지정한다.

# 이제 실제 코딩을 해보자!

```
@RestController
public class CalculatorController {

    @GetMapping("/add")
    public int addTwoNumbers(
        @RequestParam int number1,
        @RequestParam int number2
    ) {
        return number1 + number2;
    }
}
```

@RequestParam :

주어지는 쿼리를 함수 파라미터에 넣는다.

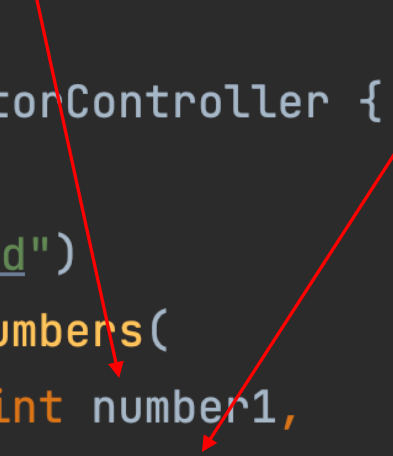
# 이제 실제 코딩을 해보자!

GET /add?number1=10&number2=20

같은 이름을 가진  
쿼리의 값이 들어온다.

```
@RestController
public class CalculatorController {

    @GetMapping("/add")
    public int addTwoNumbers(
        @RequestParam int number1,
        @RequestParam int number2
    ) {
        return number1 + number2;
    }
}
```

A diagram with two red arrows. One arrow starts from the 'number1' parameter in the URL 'GET /add?number1=10&number2=20' and points to the '@RequestParam int number1' parameter in the Java code. The other arrow starts from the 'number2' parameter in the URL and points to the '@RequestParam int number2' parameter in the Java code.

# Controller에서 @RequestParam을 제거!

```
@RestController
public class CalculatorController {

    @GetMapping("/add")
    public int addTwoNumbers(CalculatorAddRequest request) {
        return request.getNumber1() + request.getNumber2();
    }
}
```

# Controller에서 @RequestParam을 제거!

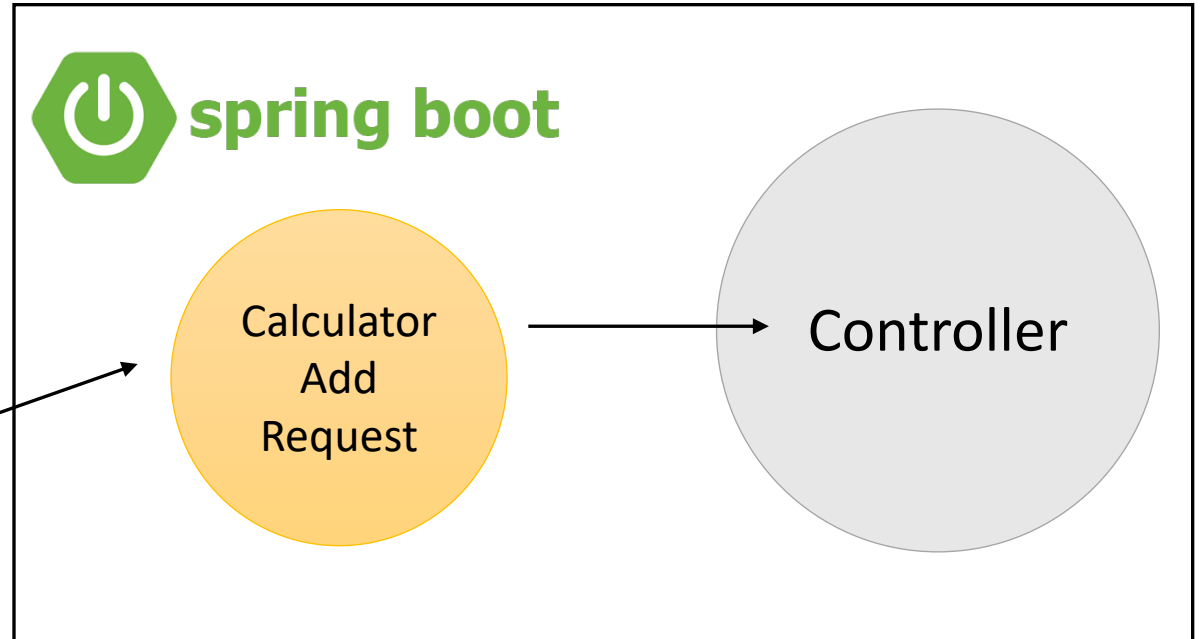
```
@RestController
public class CalculatorController {

    @GetMapping("/add")
    public int addTwoNumbers(CalculatorAddRequest request) {
        return request.getNumber1() + request.getNumber2();
    }
}
```

# Controller에서 @RequestParam을 제거!



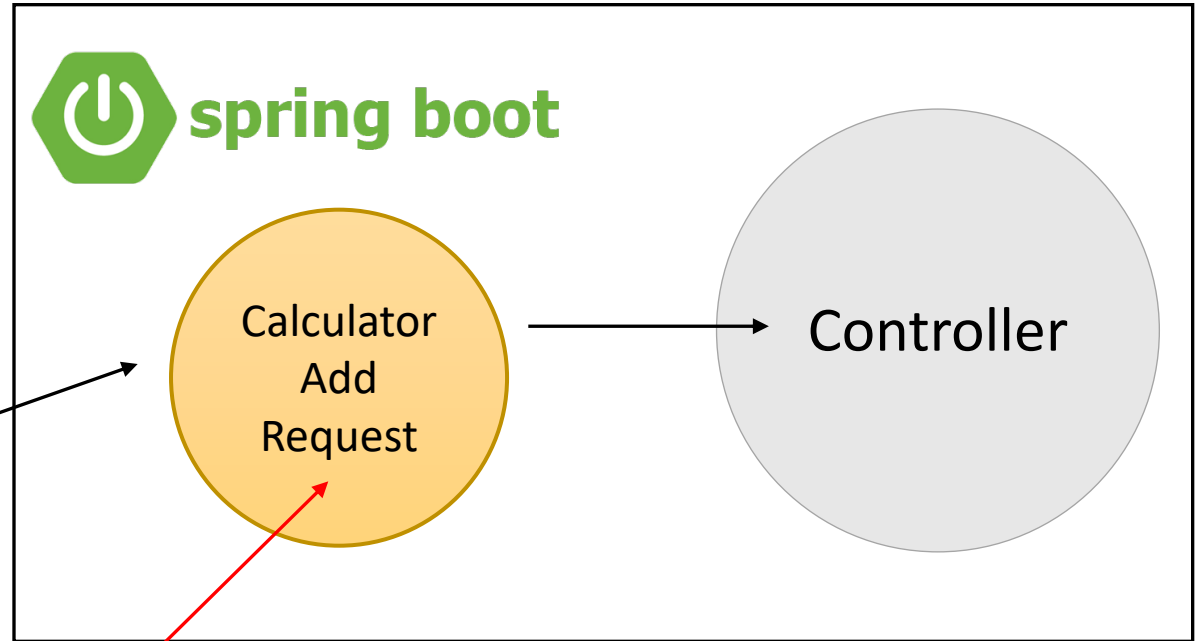
? number1=10&number2=20



# Controller에서 @RequestParam을 제거!



? number1=10&number2=20



Data Transfer Object, DTO라고 한다!



# HTTP 메시지 요청과 응답을 주고 받는다!

GET /add? number1=10&number2=20  
Host: localhost:8080



# HTTP 메시지 요청과 응답을 주고 받는다!

```
GET /add? number1=10&number2=20  
Host: localhost:8080
```



# HTTP 메시지 요청과 응답을 주고 받는다!



GET /add? number1=10&number2=20  
Host: localhost:8080



200 OK  
Content-Type: application/json  
  
30

# HTTP 메시지 요청과 응답을 주고 받는다!



200 OK  
Content-Type: application/json

30

GET /add? number1=10&number2=20  
Host: localhost:8080



# 6강. POST API 개발하고 테스트하기

**POST에서는 데이터를 어떻게 받을까?**

# GET API에서 데이터를 받을 때

쿼리 : ?number1=10&number2=20

# POST에서는 데이터를 어떻게 받을까?

HTTP Body를 이용한다!



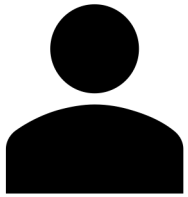
# POST에서 Body로 데이터를 어떻게 받을까?

JavaScript Object Notation, JSON

# JSON이란?!

객체 표기법, 즉 무언가를 표현하기 위한 형식이다!

# JSON이란?!



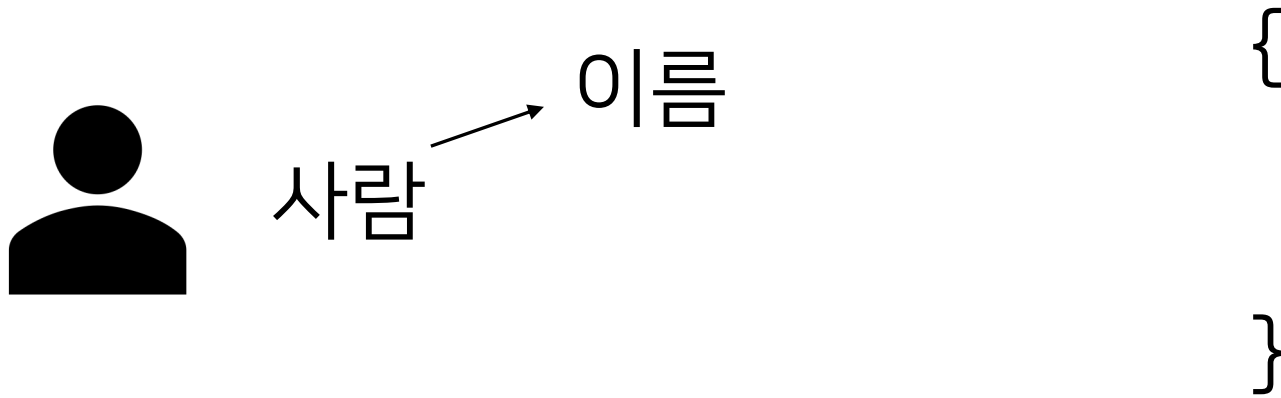
사람

{

}

JSON 표기는 중괄호가 양 끝에 있다!

# JSON이란?!



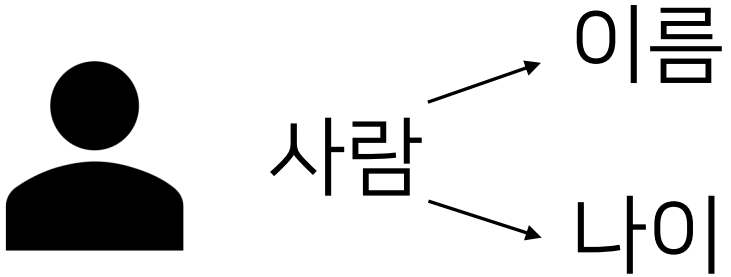
이름 = 최태현

# JSON이란?!



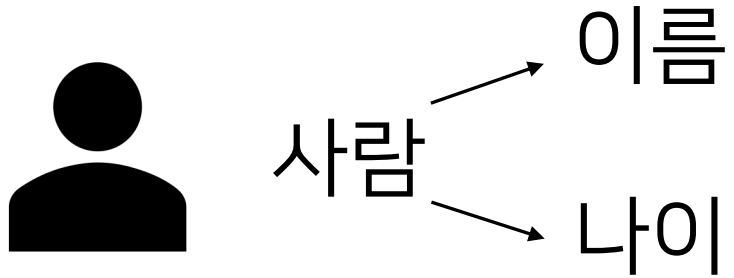
중괄호 안에, "key": value로 표기한다!

# JSON이란?!



```
{  
  "name": "최태현",  
  "age": 99  
}
```

# JSON이란?!



```
{  
  "name": "최태현",  
  "age": 99  
}
```

'속성' 각각은 심표로 구분한다!

# JSON이란?!

Java로 비유해보자면, Map<Object, Object> 느낌!



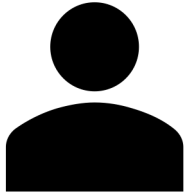
# JSON이란?!

Java로 비유해보자면, Map<Object, Object> 느낌!

Object는 정말 다양하다!



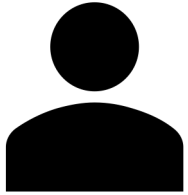
# JSON의 value에는 List가 올 수도 있다!



```
{  
  "name": "최태현",  
  "age": 99,  
  "dogs": ["코코", "초코"]  
}
```

강아지들을 키운다면?!

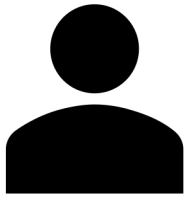
# JSON의 value에는 다른 JSON이 올 수도 있다!



```
{  
  "name": "최태현",  
  "age": 99  
}
```

집 정보를 추가하고 싶다면?!

# JSON의 value에는 다른 JSON이 올 수도 있다!



```
{  
  "name": "최태현",  
  "age": 99,  
  "house": {  
    "address": "대한민국 서울",  
    "hasDoor": true  
  }  
}
```

집 정보를 추가하고 싶다면?!

**POST에서 Body로 데이터를 어떻게 받을까?**

# GET은 이렇게 했었죠!!

GET /add? number1=10&number2=20  
Host: localhost:8080



# POST에서 Body로 데이터를 어떻게 받을까?

```
POST/multiply  
Host: localhost:8080
```

```
{  
  "number1": 10,  
  "number2": 20  
}
```



# POST에서 Body로 데이터를 어떻게 받을까?

```
POST/ multiply  
Host: localhost:8080
```

```
{  
  "number1": 10,  
  "number2": 20  
}
```



POSTMAN

HTTP 요청 바디



spring boot



**그럼 이제 곱셈 기능을 POST API로 만들어 봅시다!**

# 곱셈 API

HTTP Method

HTTP Path

HTTP Body (JSON)

API의 반환 결과

# 곱셈 API

HTTP Method          POST

HTTP Path

HTTP Body (JSON)

API의 반환 결과

# 곱셈 API

HTTP Method

POST

HTTP Path



/multiply

HTTP Body (JSON)

API의 반환 결과

# 곱셈 API

HTTP Method

POST

HTTP Path

/multiply

HTTP Body (JSON) 

```
{  
  "number1": 숫자,  
  "number2": 숫자  
}
```

API의 반환 결과

# 곱셈 API

HTTP Method

POST

HTTP Path

/multiply

HTTP Body (JSON)

```
{  
  "number1": 숫자,  
  "number2": 숫자  
}
```

API의 반환 결과

숫자 (곱셈 결과)



**이제 코딩하러 가보시죠~!!**

# 만들어진 API

```
@PostMapping(🌐📄"/multiply")  
public int multiplyTwoNumbers(@RequestBody CalculatorMultiplyRequest request) {  
    return request.getNumber1() * request.getNumber2();  
}
```



# 만들어진 API

```
@PostMapping(🌐"/multiply")  
public int multiplyTwoNumbers(@RequestBody CalculatorMultiplyRequest request) {  
    return request.getNumber1() * request.getNumber2();  
}
```

한 Controller Class에 여러 API를 추가할 수 있다!

# 만들어진 API

```
@PostMapping(🌐📄"/multiply")
```

```
public int multiplyTwoNumbers(@RequestBody CalculatorMultiplyRequest request) {  
    return request.getNumber1() * request.getNumber2();  
}
```

@PostMapping("/multiply")

아래 함수를 HTTP Method가 POST이고  
Path가 /multiply인 API로 만든다!

# 만들어진 API

```
@PostMapping(🌐"/multiply")
public int multiplyTwoNumbers(@RequestBody CalculatorMultiplyRequest request) {
    return request.getNumber1() * request.getNumber2();
}
```


## @RequestBody

HTTP Body로 들어오는 JSON을  
CalculatorMultiplyRequest로 바꾼다!

# 이름이 같아야 합니다!

```
{  
  "number1": 숫자,  
  "number2": 숫자  
}
```

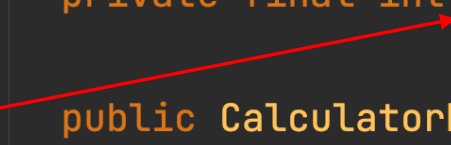
```
public class CalculatorMultiplyRequest {  
  
    2 usages  
    private final int number1;  
  
    2 usages  
    private final int number2;  
  
    public CalculatorMultiplyRequest(int number1, int number2) {  
        this.number1 = number1;  
        this.number2 = number2;  
    }  
  
    1 usage  
    public int getNumber1() { return number1; }  
  
    1 usage  
    public int getNumber2() { return number2; }  
  
}
```



# 이름이 같아야 합니다!

```
{  
  "number1": 숫자,  
  "number2": 숫자  
}
```

```
public class CalculatorMultiplyRequest {  
  
    2 usages  
    private final int number1;  
  
    2 usages  
    private final int number2;  
  
    public CalculatorMultiplyRequest(int number1, int number2) {  
        this.number1 = number1;  
        this.number2 = number2;  
    }  
  
    1 usage  
    public int getNumber1() { return number1; }  
  
    1 usage  
    public int getNumber2() { return number2; }  
  
}
```



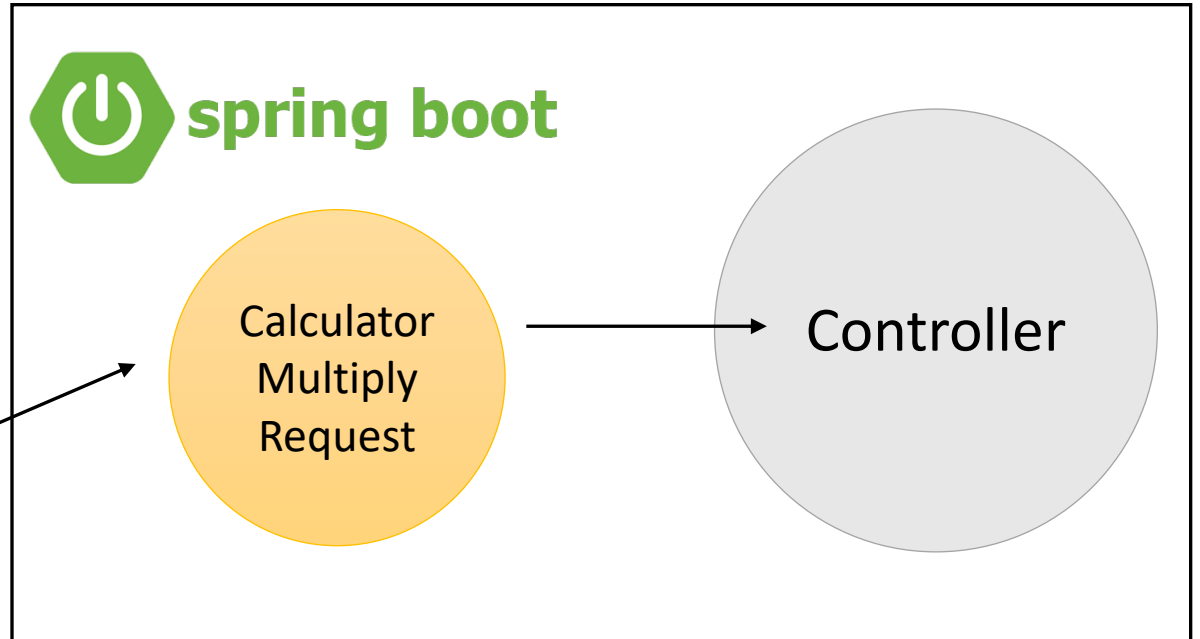
**POST MAN으로 테스트 해보자!**

# HTTP Body는 CalculatorMultiplyRequest에 매핑된다!



POST/ multiply  
Host: localhost:8080

```
{  
  "number1": 10,  
  "number2": 15  
}
```



# HTTP 메시지 요청과 응답을 주고 받는다!



POST/ multiply  
Host: localhost:8080

```
{  
  "number1": 10,  
  "number2": 15  
}
```





# HTTP 메시지 요청과 응답을 주고 받는다!



```
POST/ multiply  
Host: localhost:8080
```

```
{  
  "number1": 10,  
  "number2": 15  
}
```

# HTTP 메시지 요청과 응답을 주고 받는다!



200 OK  
Content-Type: application/json  
150



# HTTP 메시지 요청과 응답을 주고 받는다!

```
200 OK  
Content-Type: application/json  
  
150
```



# 7강. 유저 생성 API 개발

# 도서관리 애플리케이션의 요구사항

## 사용자

- 도서관의 사용자를 등록할 수 있다. (이름 필수, 나이 선택)
- 도서관 사용자의 목록을 볼 수 있다.
- 도서관 사용자 이름을 업데이트 할 수 있다.
- 도서관 사용자를 삭제할 수 있다.

## 책

- 도서관에 책을 등록 및 삭제할 수 있다.
- 사용자가 책을 빌릴 수 있다.
  - 다른 사람이 그 책을 진작 빌렸다면 빌릴 수 없다.
- 사용자가 책을 반납할 수 있다.

# 미리 만들어져 있는 웹 UI

<http://localhost:8080/v1/index.html>



### 사용자 등록

이름

나이

저장



### 책 등록

책 이름

저장



### 책 대출

이름

책 이름

저장



### 책 반납

이름

책 이름

저장

# 도서관 사용자를 등록할 수 있다 (이름 필수, 나이 선택)

가장 먼저 API를 설계해야 한다!



# 도서관 사용자를 등록할 수 있다 (이름 필수, 나이 선택)

- HTTP Method : POST
- HTTP Path : /user
- HTTP Body (JSON)

```
{  
  "name": String (null 불가능),  
  "age": Integer  
}
```

- 결과 반환 X (HTTP 상태 200 OK이면 충분하다)

# 8강. 유저 조회 API 개발과 테스트

# 유저 조회 API 스펙

- HTTP Method : GET
- HTTP Path : /user
- 쿼리 : 없음
- 결과 반환

JSON ▾

```
[{  
  "id": Long,  
  "name": String (null 불가능),  
  "age": Integer  
}, ...]
```

# 1) 결과 반환이 JSON이다?!

Controller에서 getter가 있는 객체를 반환하면 JSON이 된다!

# 1) 결과 반환이 JSON이다?!

```
public class Fruit {  
  
    public String getName() {  
        return name;  
    }  
  
    public long getPrice() {  
        return price;  
    }  
}
```

```
1 {  
2   "name": "사과",  
3   "price": 1000  
4 }
```

## 2) Id는 무엇인가?!

- HTTP Method : GET
- HTTP Path : /user
- 쿼리 : 없음
- 결과 반환

JSON ▾

```
[{  
  "id": Long,  
  "name": String (null 불가능),  
  "age": Integer  
}, ...]
```

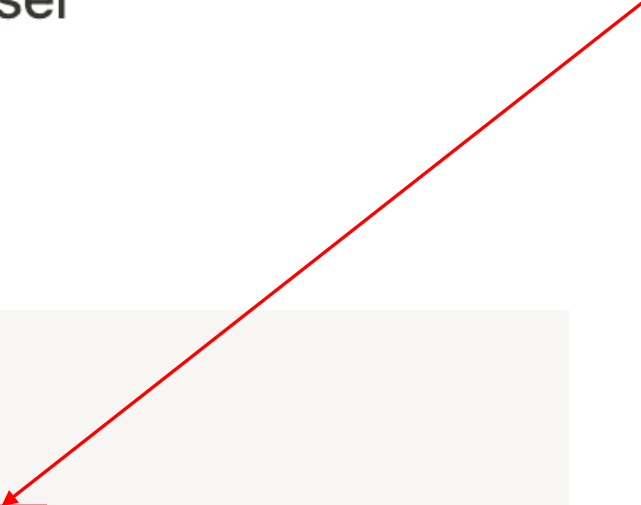
## 2) Id는 무엇인가?!

- HTTP Method : GET
- HTTP Path : /user
- 쿼리 : 없음
- 결과 반환

유저별로 겹치지 않는 유일한 번호

JSON ▾

```
[{  
  "id": Long,  
  "name": String (null 불가능),  
  "age": Integer  
}, ...]
```



## 2) Id는 무엇인가?!

List에 담겨 있는 유저의 순서를 id로 해주자!



# 9강. Section 1 정리. 다음으로!

# 생애 최초 API 만들기

1. 스프링 프로젝트를 시작하는 방법과 실행하는 방법
2. 네트워크, IP, 도메인, 포트, HTTP 요청과 응답 구조, 클라이언트 - 서버 구조, API와 같은 기반 지식
3. Spring Boot를 이용해 GET API와 POST API를 만드는 방법

# 우리가 만든 API의 문제점

서버를 재시작하면 유저 정보가 날아간다!

# 우리가 만든 API의 문제점

유저 정보는 메모리에서만 유지되고 있기 때문

**감사합니다**