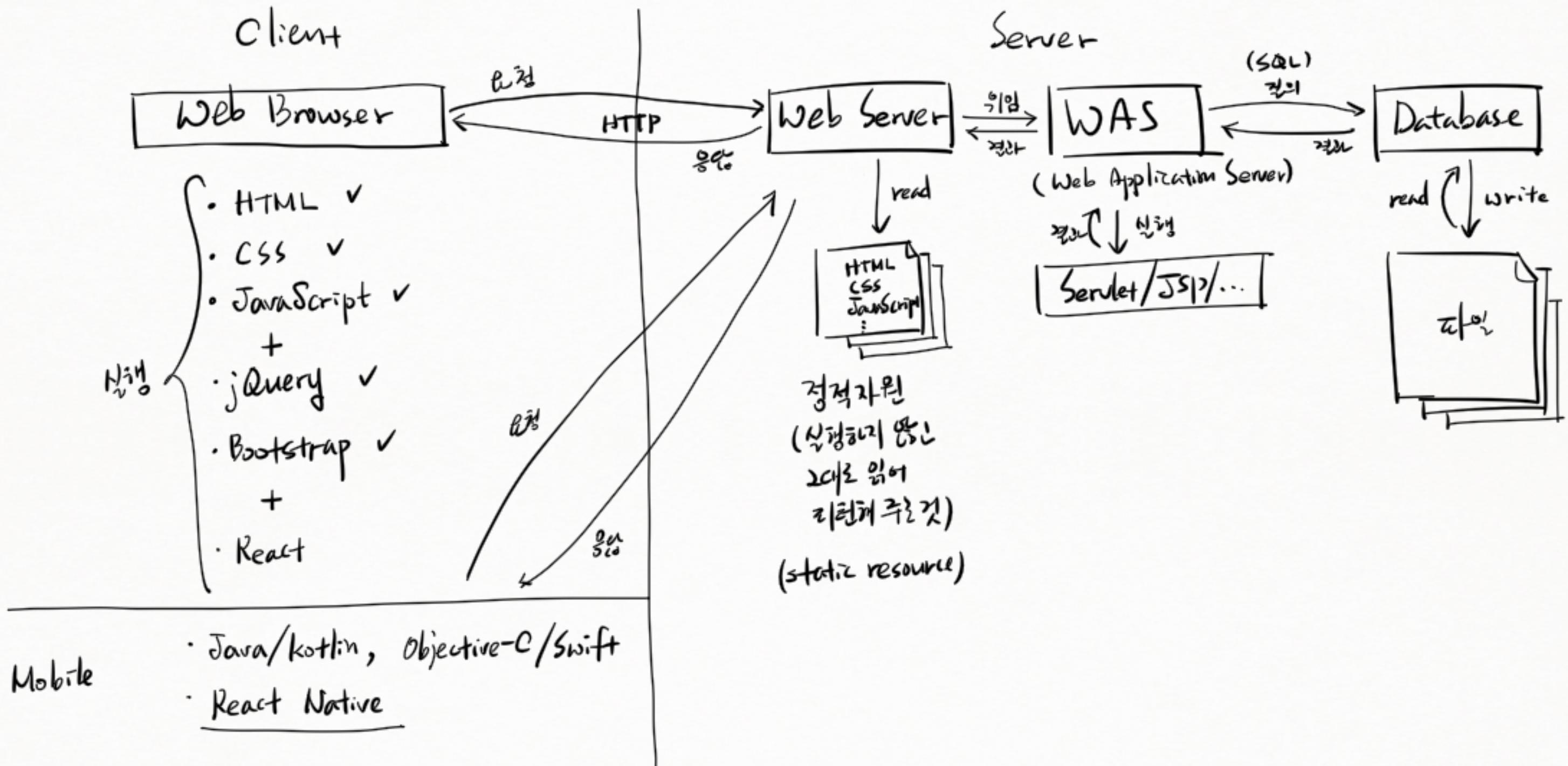
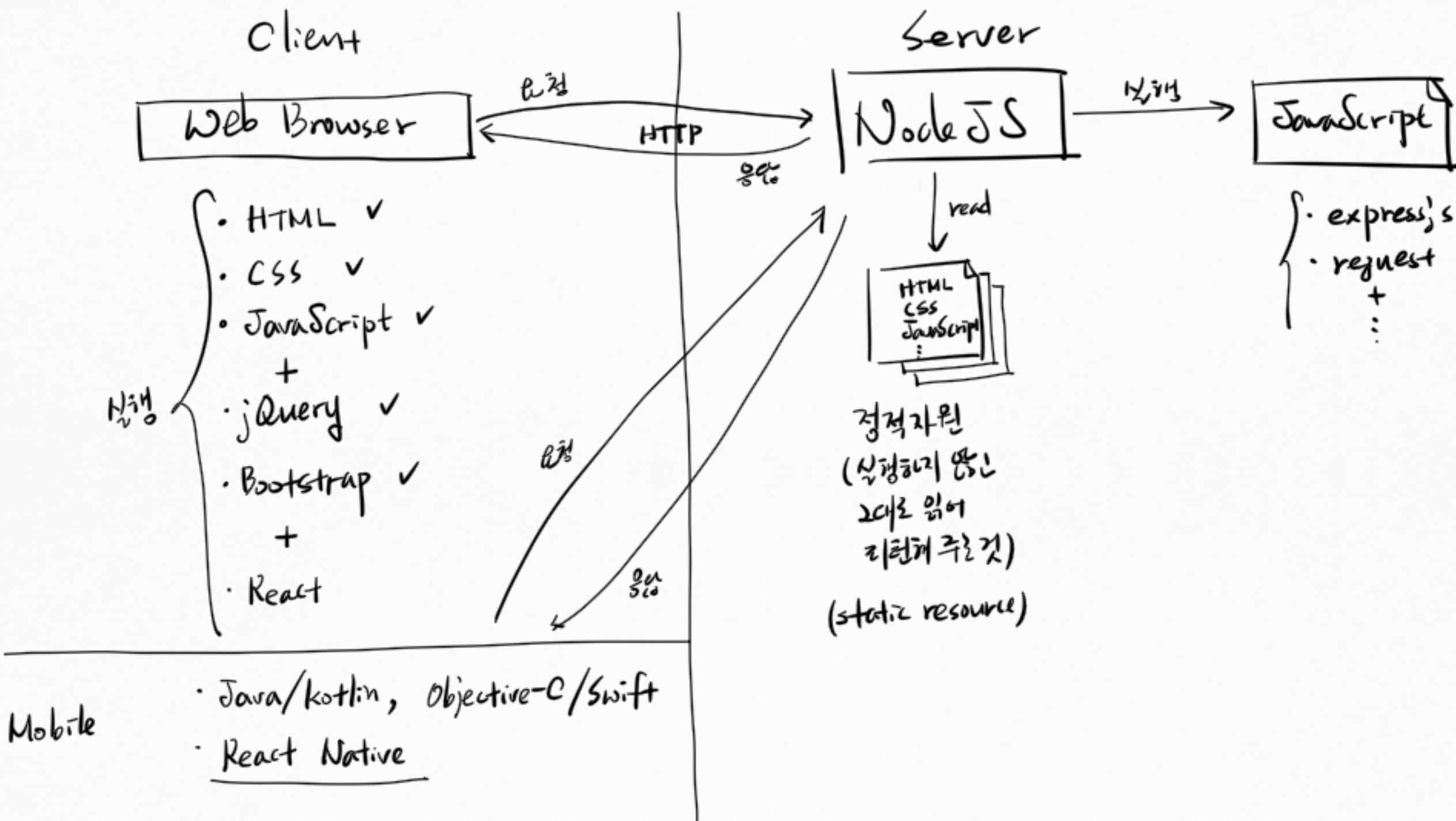


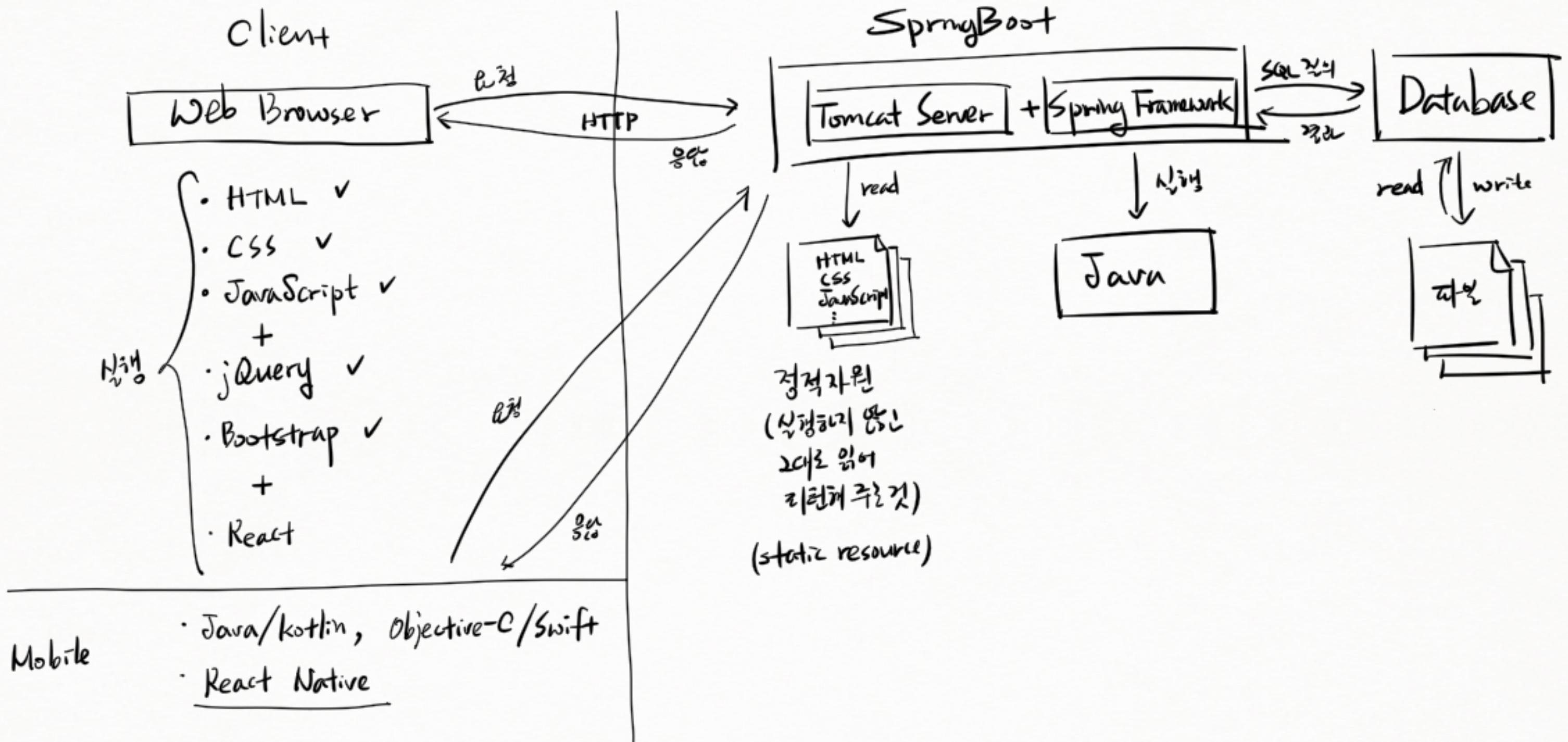
## \* Web Application Architecture by Java



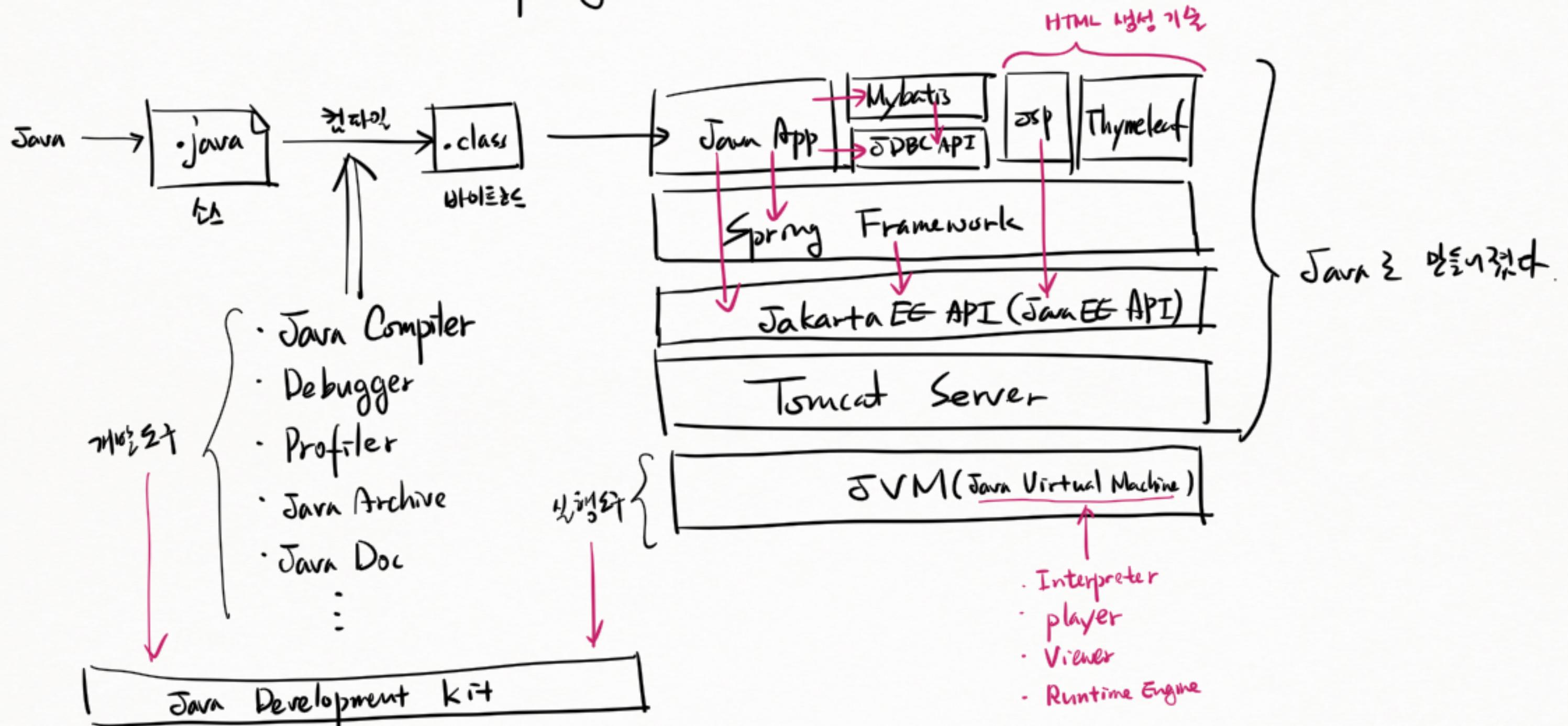
## \* Node.js Web Application Architecture



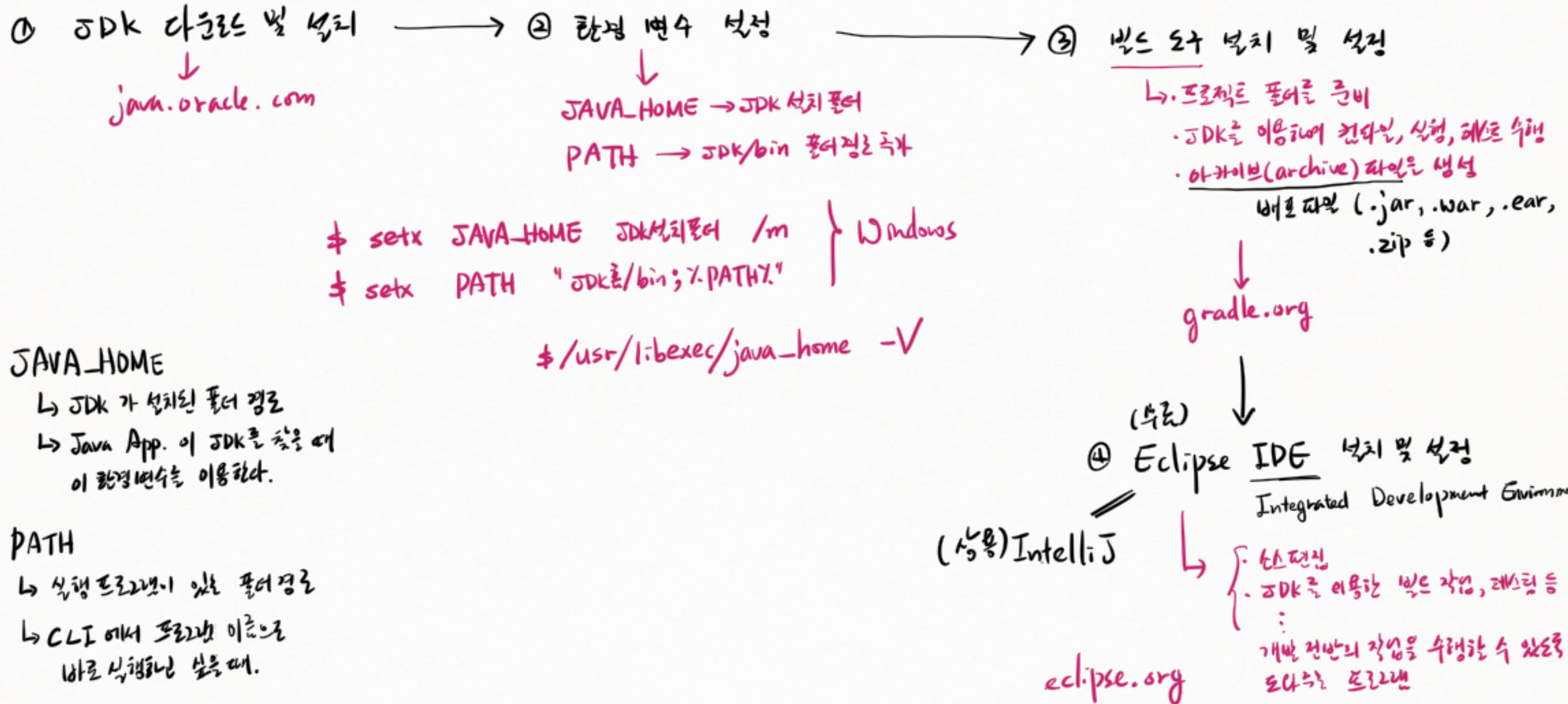
## \* SpringBoot Web Application Architecture



## \* SpringBoot 기술 스택

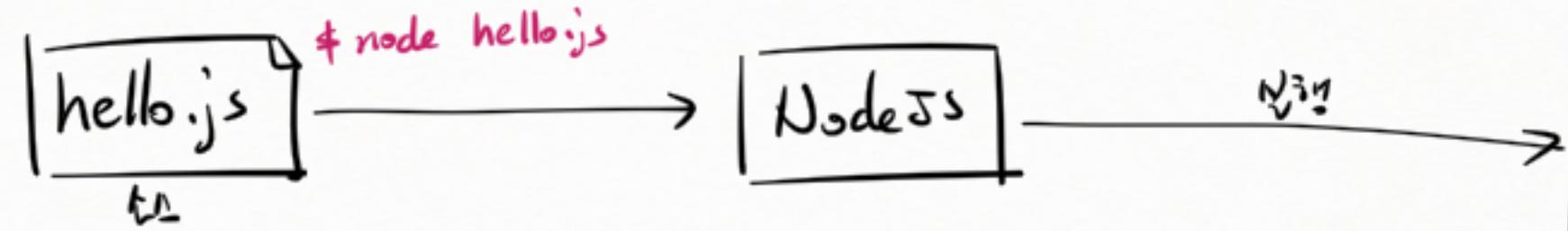


## \* Java 프로그래밍 준비

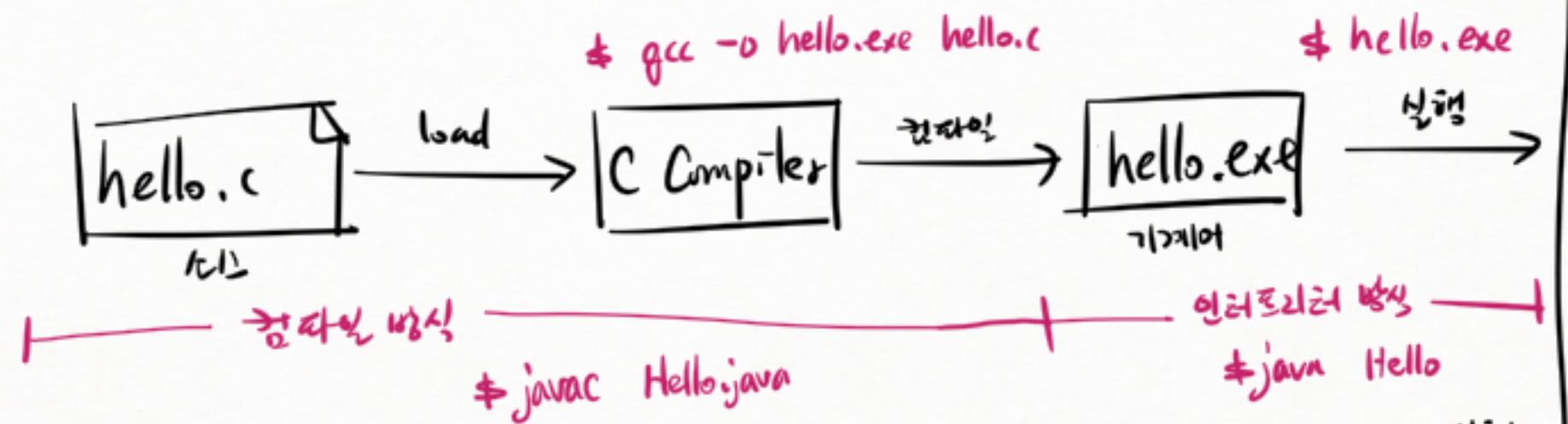


\* App. 구현 방식 및 차이

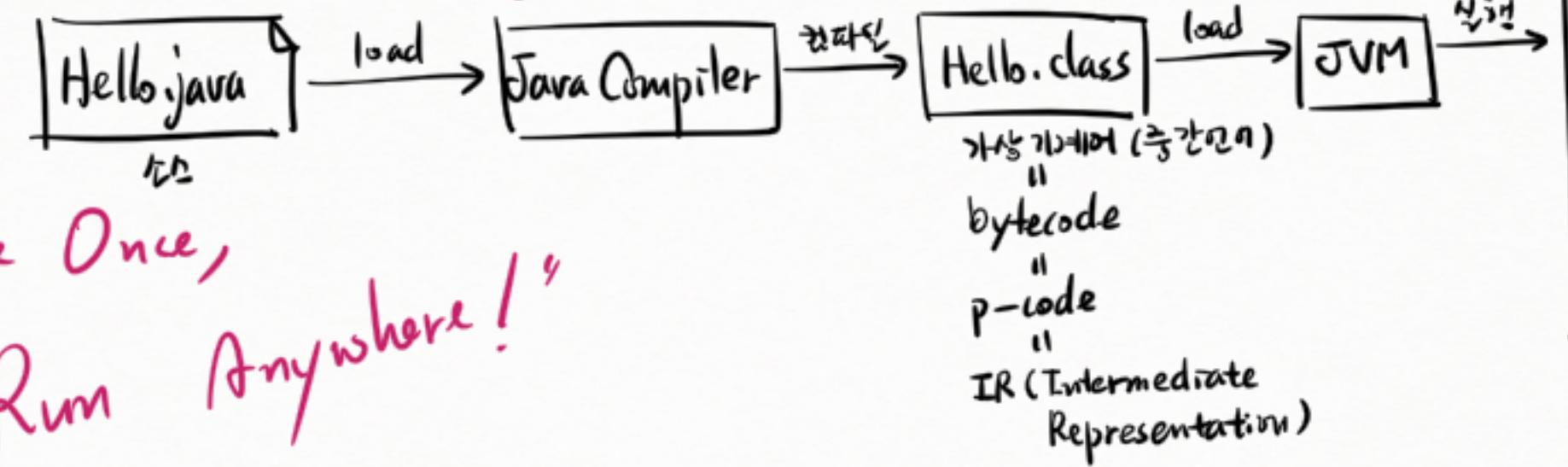
① 인터프리터 방식



② 컴파일 방식



③ 혼합형 방식  
(hybrid)



"Write Once,  
Run Anywhere!"

## \* Hybrid 방식을 도입한 이유

- 선수 깃자기 방식  
보다 나은 이유
- ① OS마다 따로 컴파일 할 필요가 없다.  
↳ 동일한 바이트코드 생성
  - ② OS용 JVM이 설치되어 있으면 실행할 수 있다.

소스



컴파일

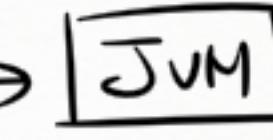
바이트코드

- 방법검사 수행 → 불법 오류를 모두 찾아낸다
- 최적화 → 실행 성능 향상



→

Windows



→

macOS



→

Linux

바이트코드 툴리티

- 인터프리터
- 전파망 인진
- 베타일 머신

실행할 컴퓨터  
불법 오류 검사를  
하는

선수 인터프리터 방식보다  
수행 속도가 빠르다.

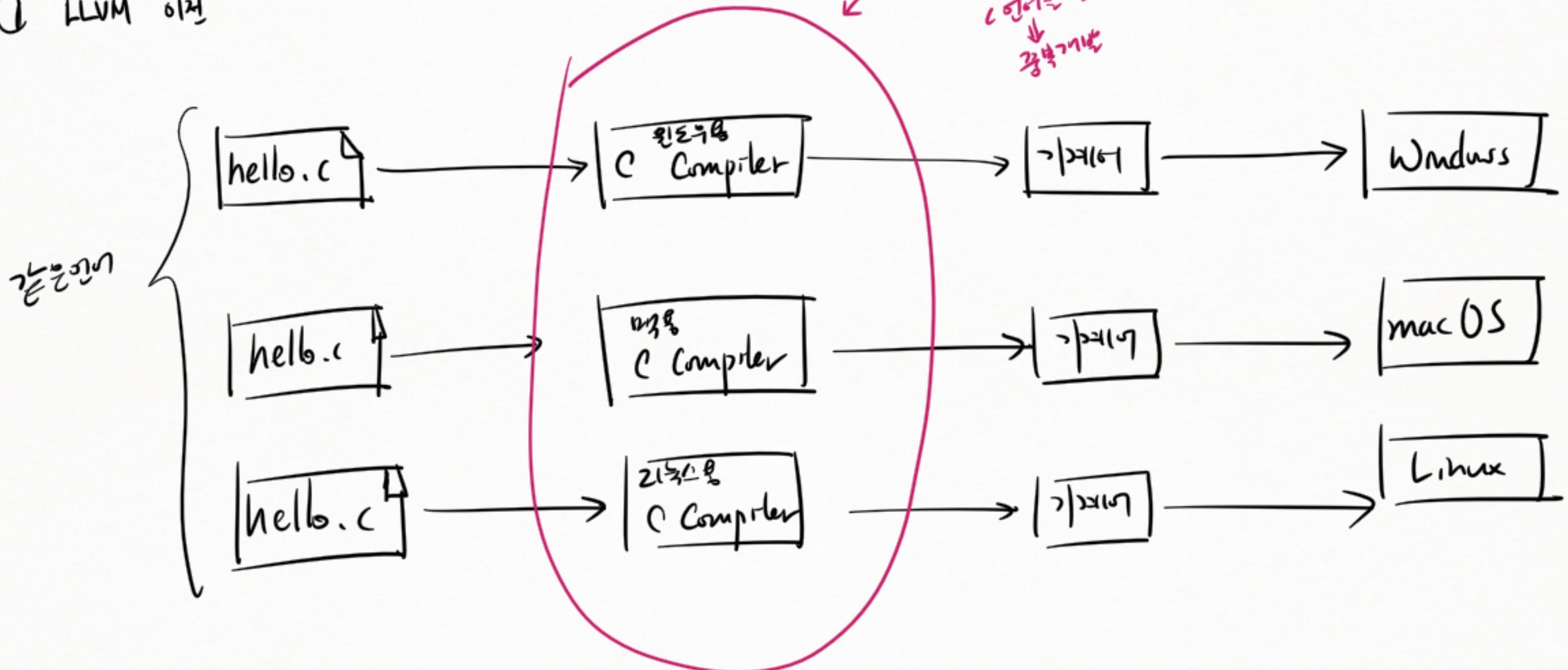
선수 인터프리터 방식  
보다 나은 이유

- ③ 컴파일 과정이 실행의 오류를 모두 찾아낸다
- ④ 일반 기계어는 하드웨어 기계어에 가까운 언어로  
변환된 명령을 실행하니 대용량  
소스에 작성된 명령을 실행하는 것보다  
수행 속도가 빠르다.

\* LLVM 저작자

Low Level Virtual Machine

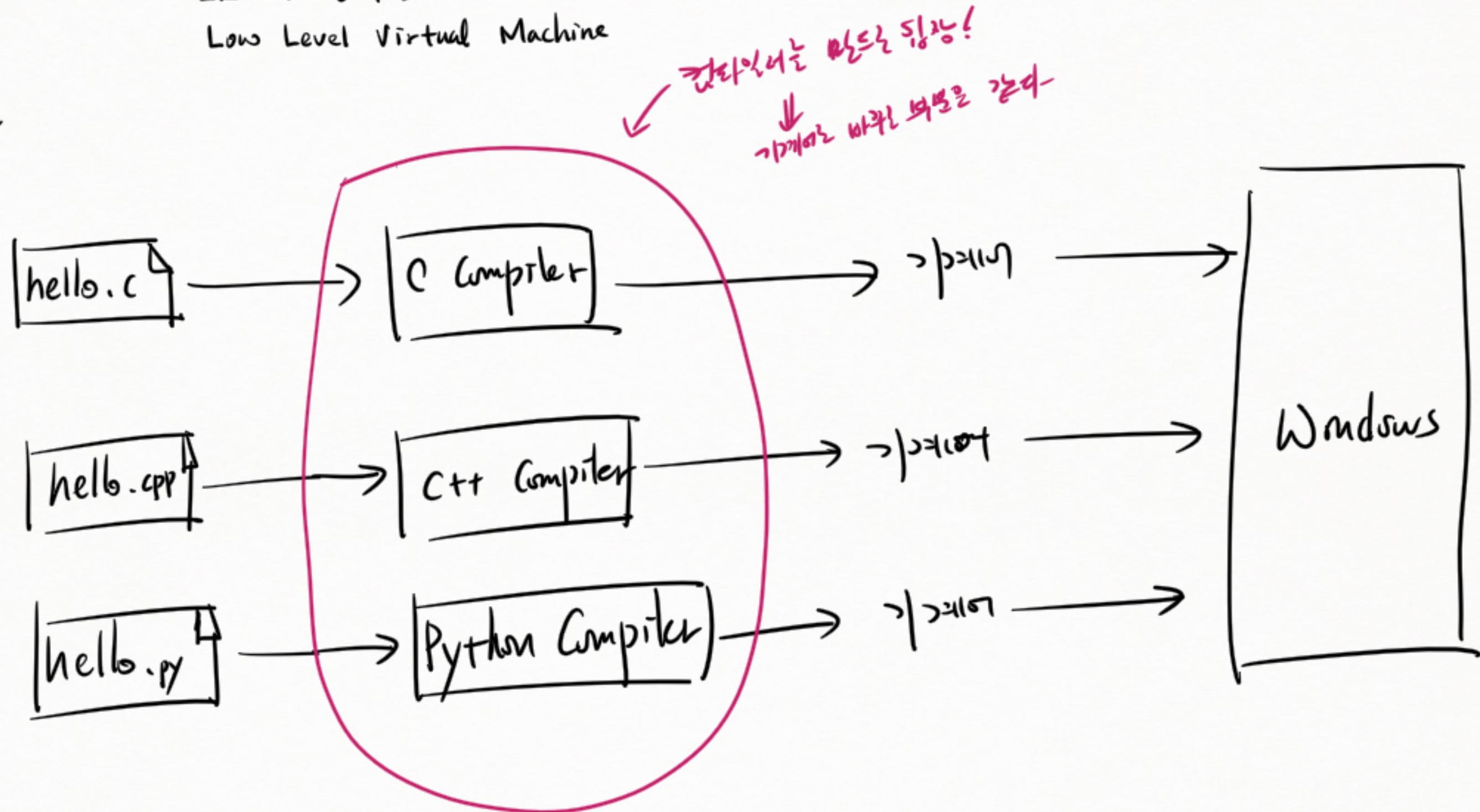
① LLVM 이전



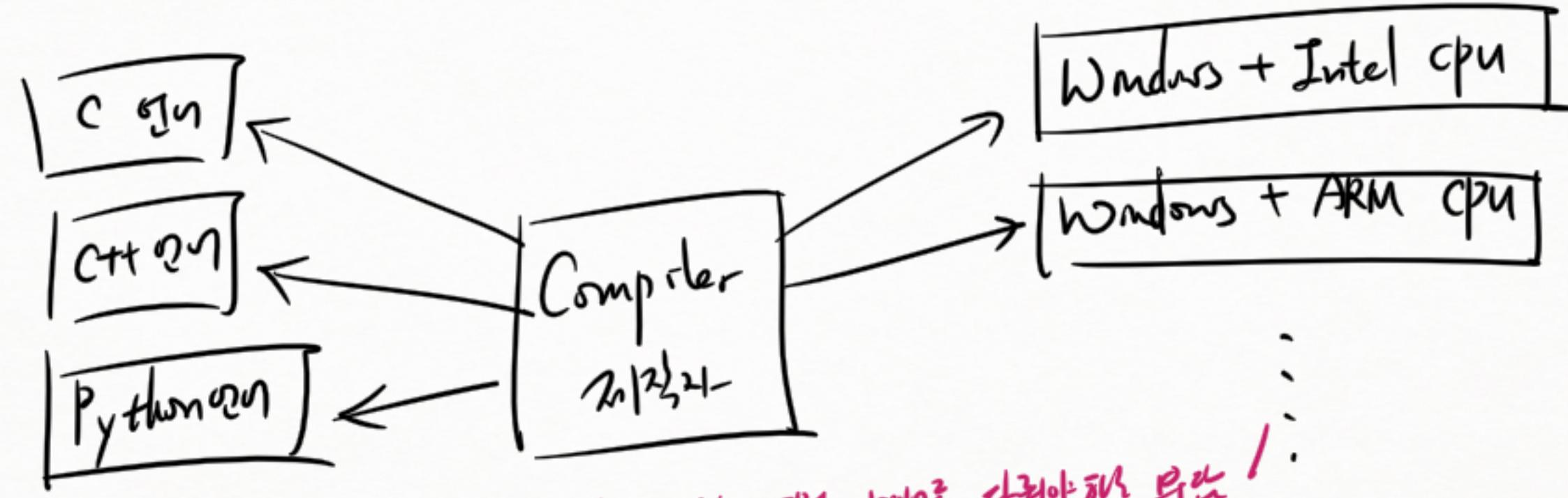
\* LLVM 저작자

Low Level Virtual Machine

① LLVM 이전

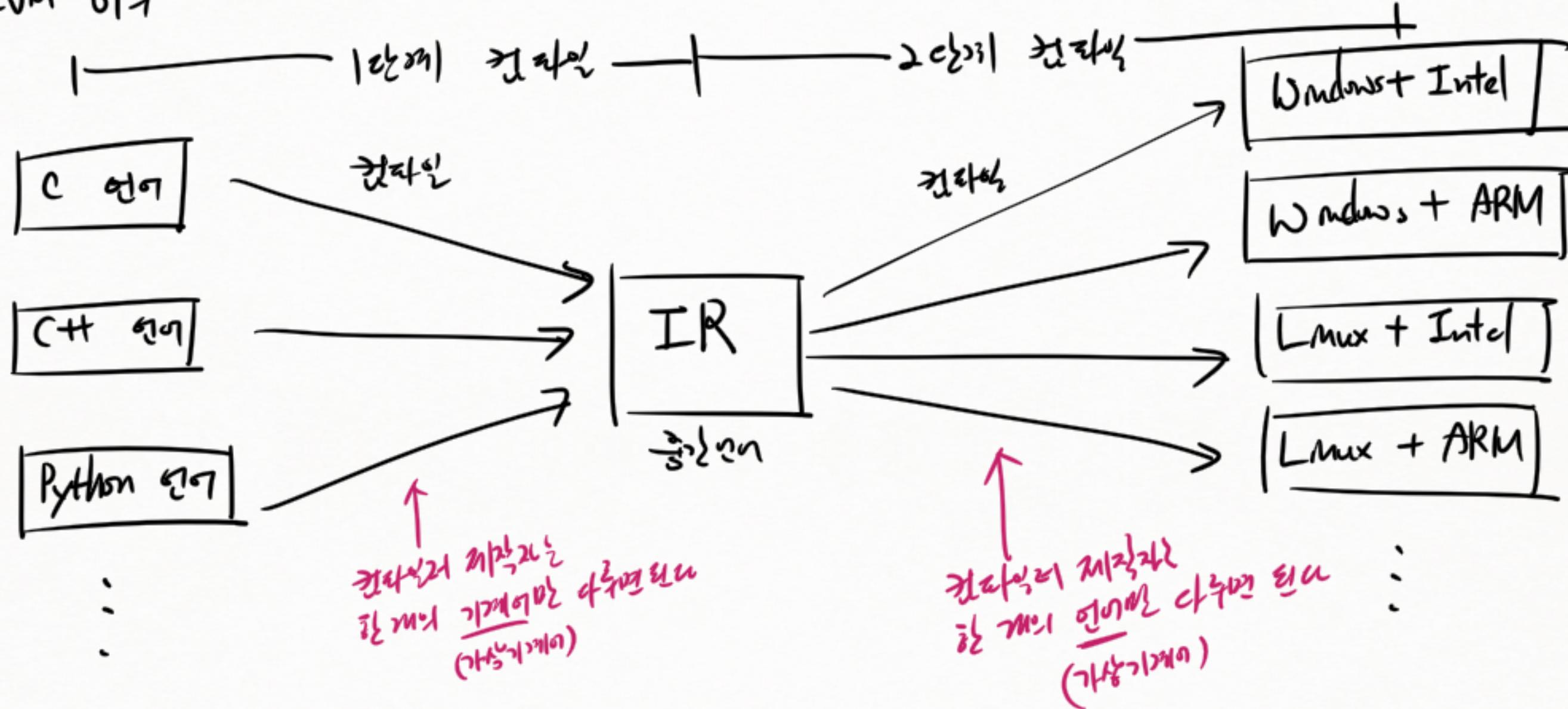


+ 가끔 틀렸을 때 예제

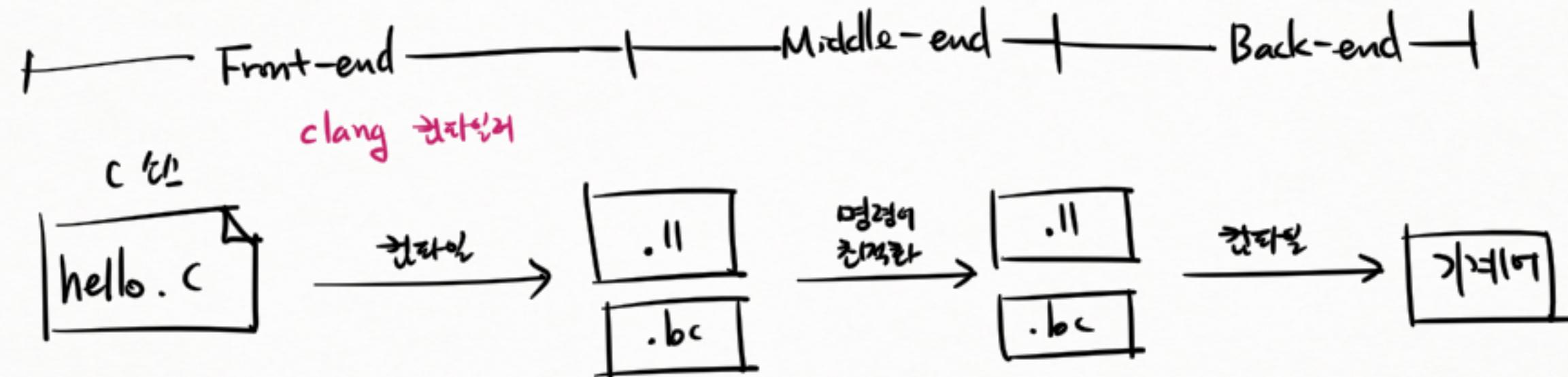


\* LLVM 10장은 사용하는 기준  
↳ 새 프로그램 언어와 컴파일러를 만들기 쉽다.

## ② LLVM 이후



## \* LLVM 컴파일 과정

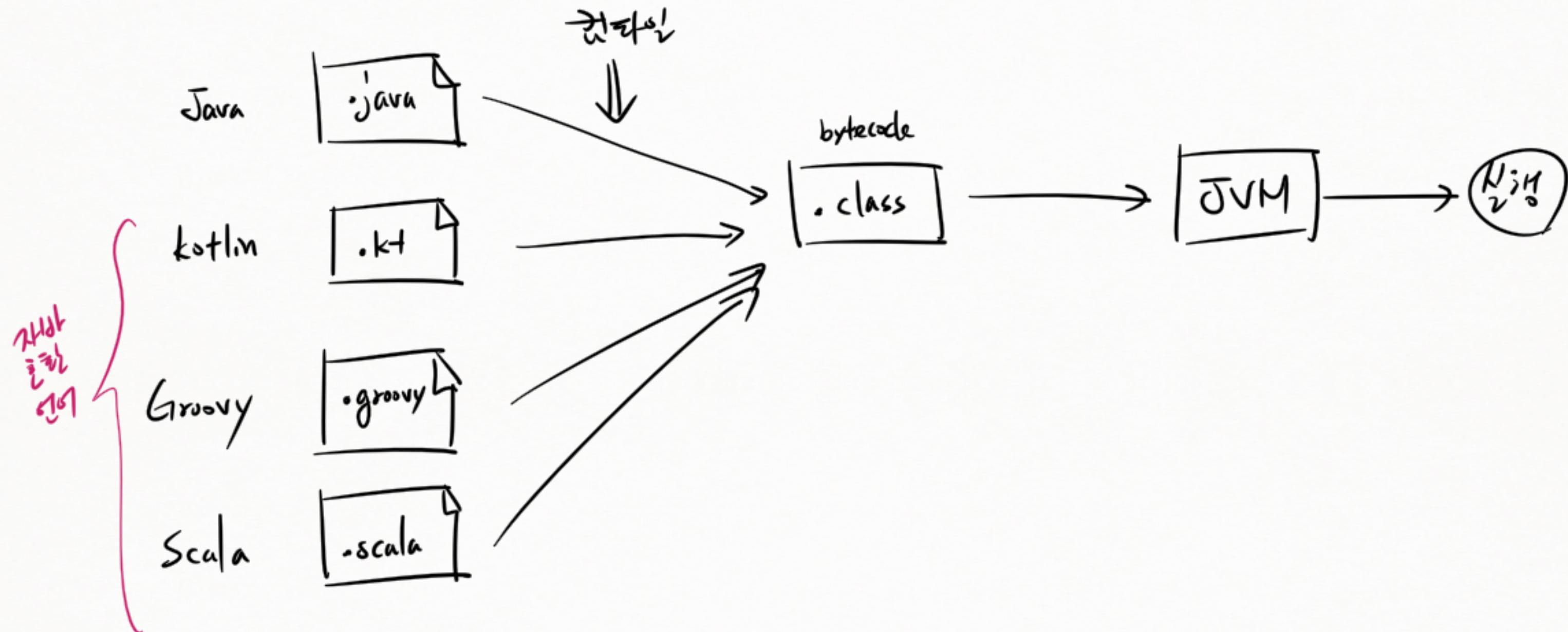


LLVM Assembly

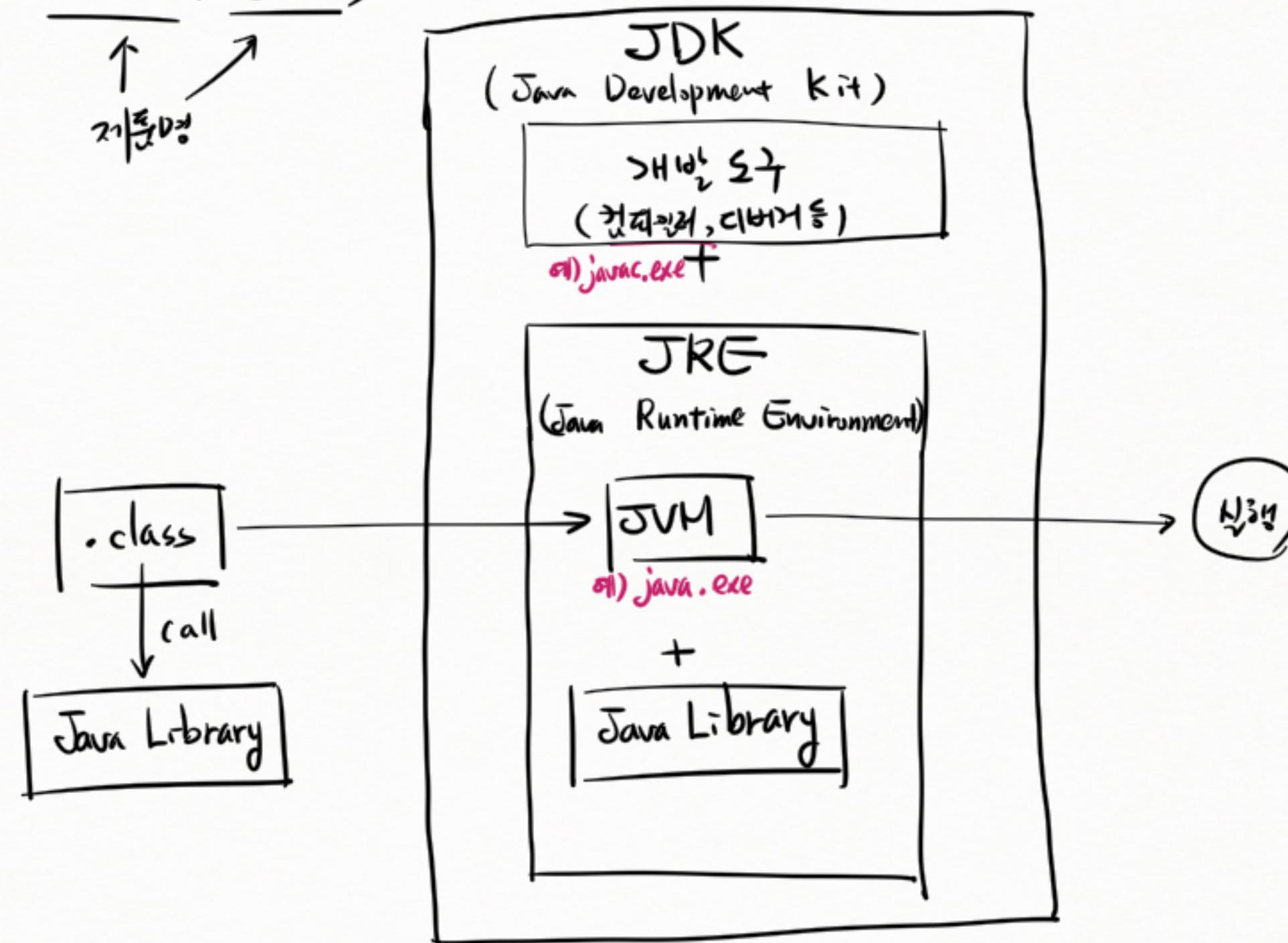
LLVM bitcode

LLVM bytecode

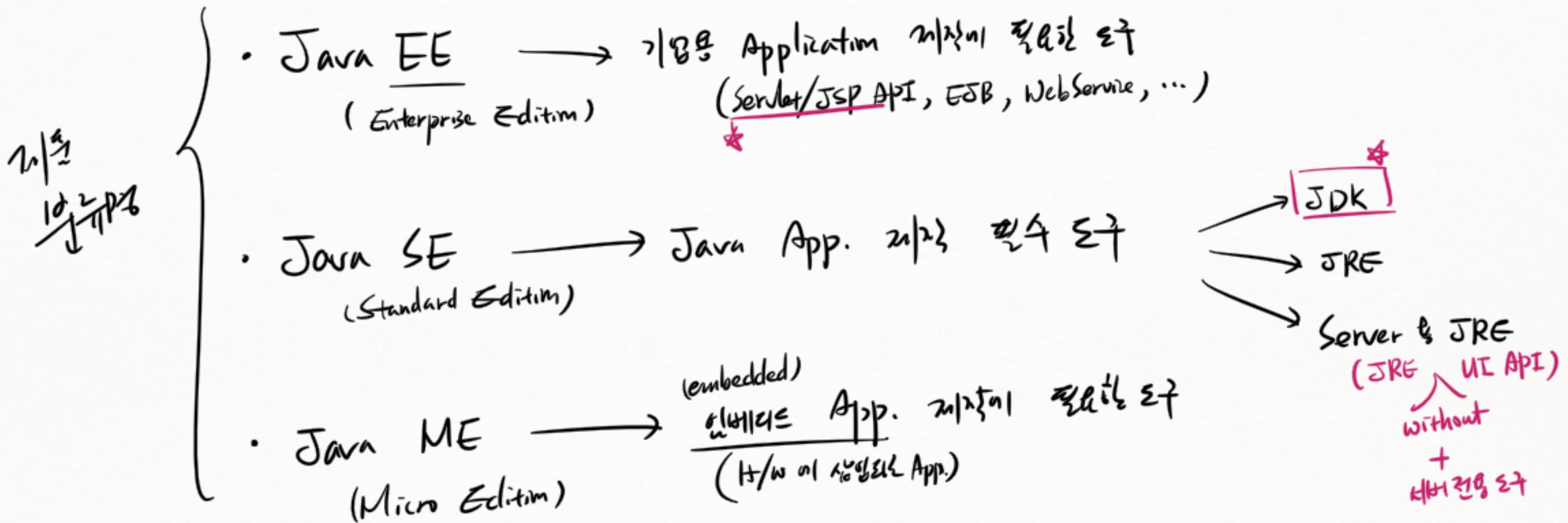
## \* Java et LLVM



## \* JDK, JRE, JVM



## \* Java EE, Java SE, Java ME



\* Eclipse IDE



+ plug-in ⇒ 개별 도구 박스

- JDT
- CDT
- :

## \* Java Project 폴더 구조

① 프로젝트 폴더 생성

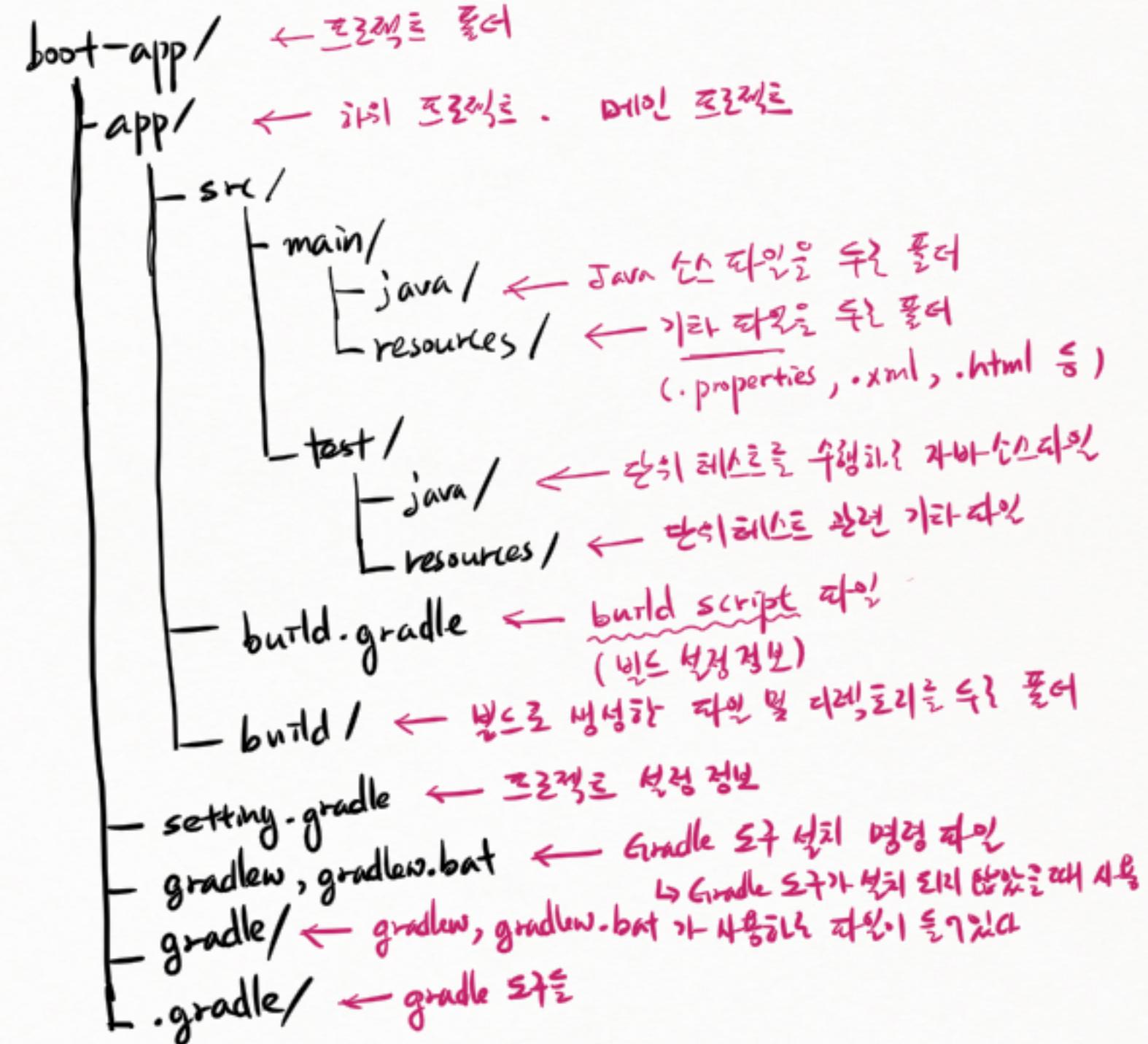
c:\Users\bitcamp\git\bitcamp-ncp\  
User Home  
↳ boot-project\

② 'boot-project\' 폴더는 Java 프로젝트 폴더로 초기화

\$ gradle init

③ 기본 이제 프로그램 실행

\$ gradle -q run



## \* SpringBoot 프로젝트 만들기

spring.io 사이트 접속 → Spring Boot 메뉴 → Spring Initializr 실행

- project: gradle-groovy

- Language: Java

- Spring Boot: 3.0.1

- Dependencies:

- ① Spring Boot Dev Tools ← 자동로딩

- ② Spring Configuration Processor ← properties 대체  
자동로딩

- ③ Spring Web

## \* Java 와서 Spring Boot 까지

