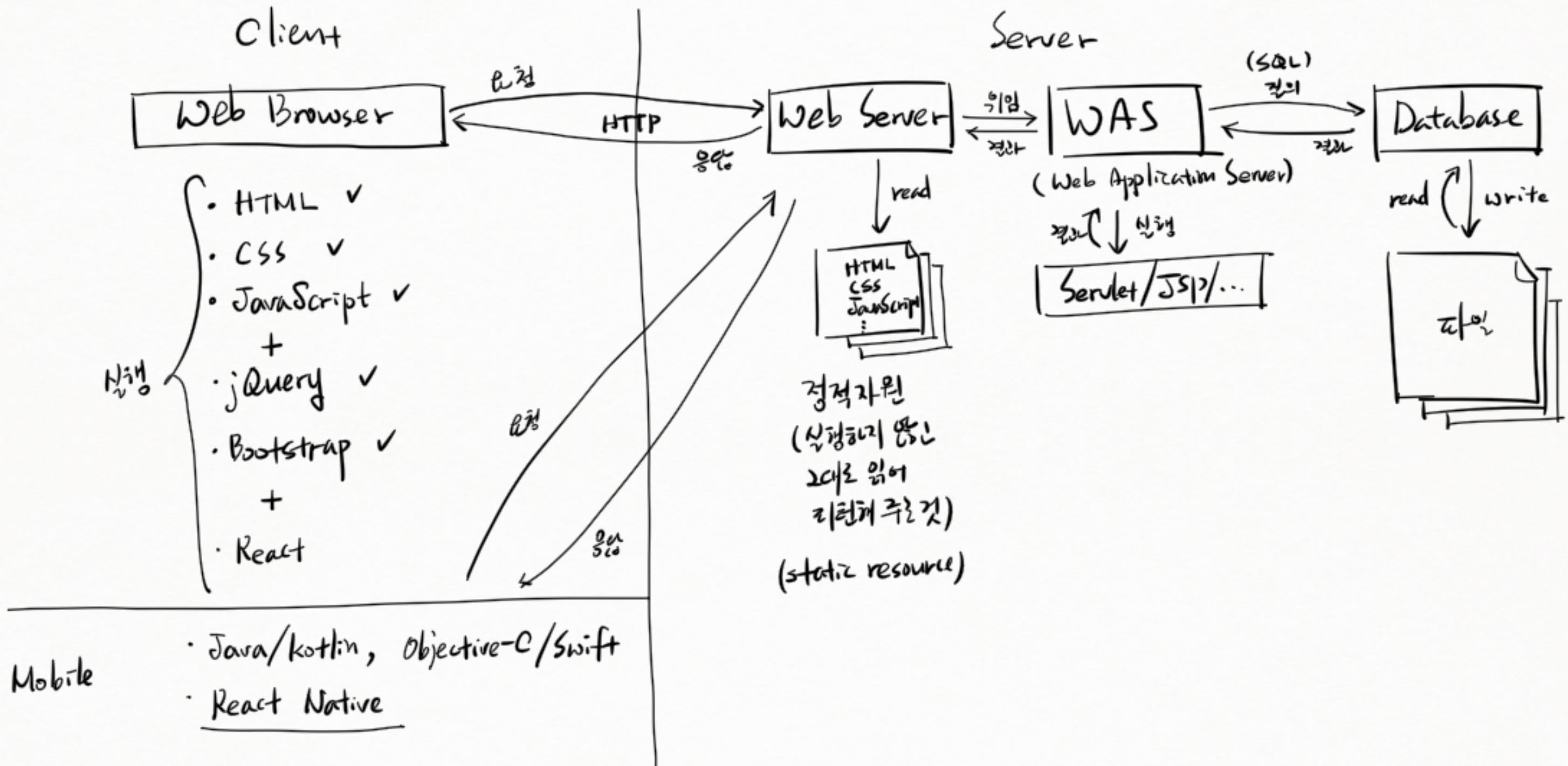
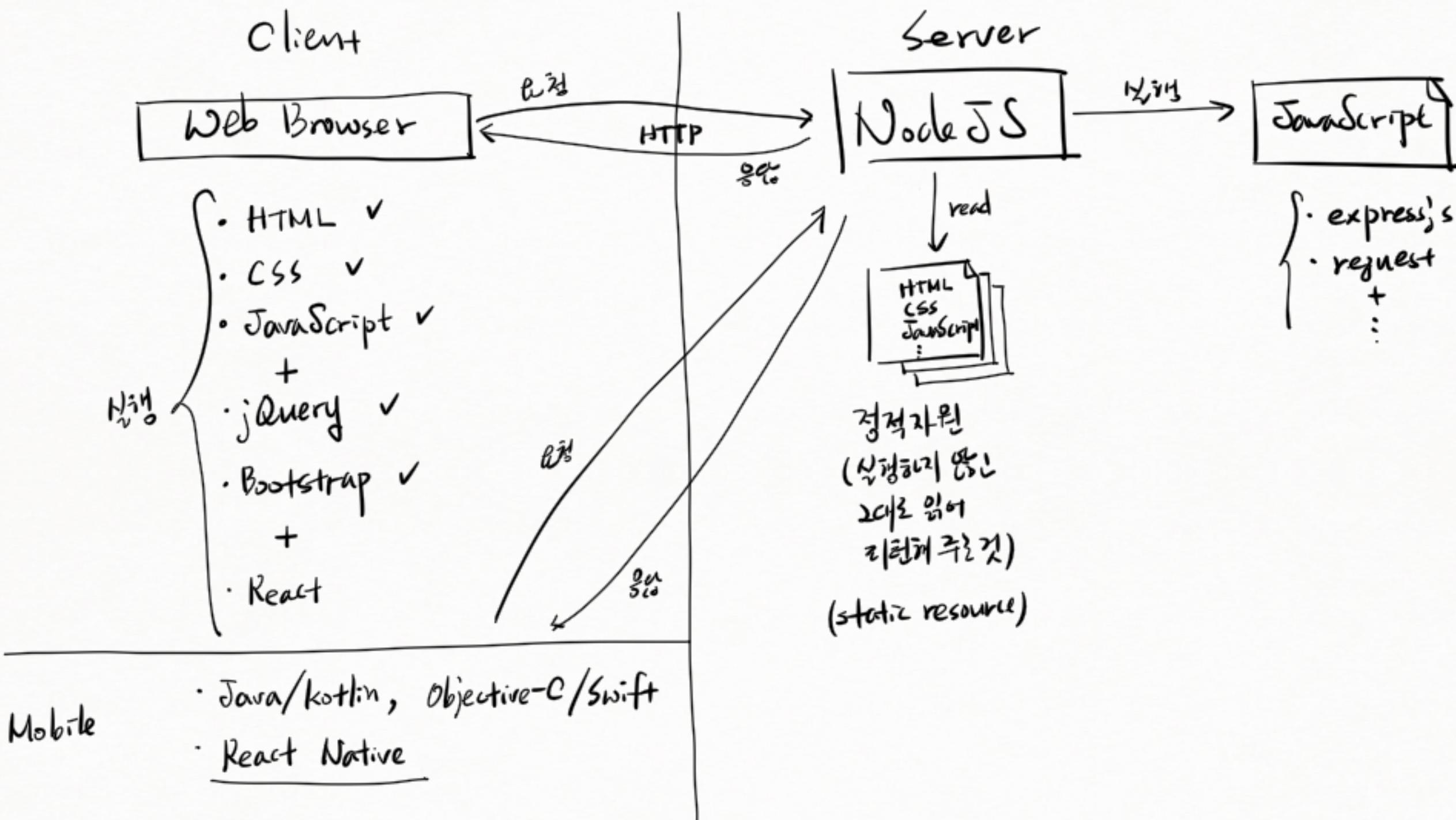


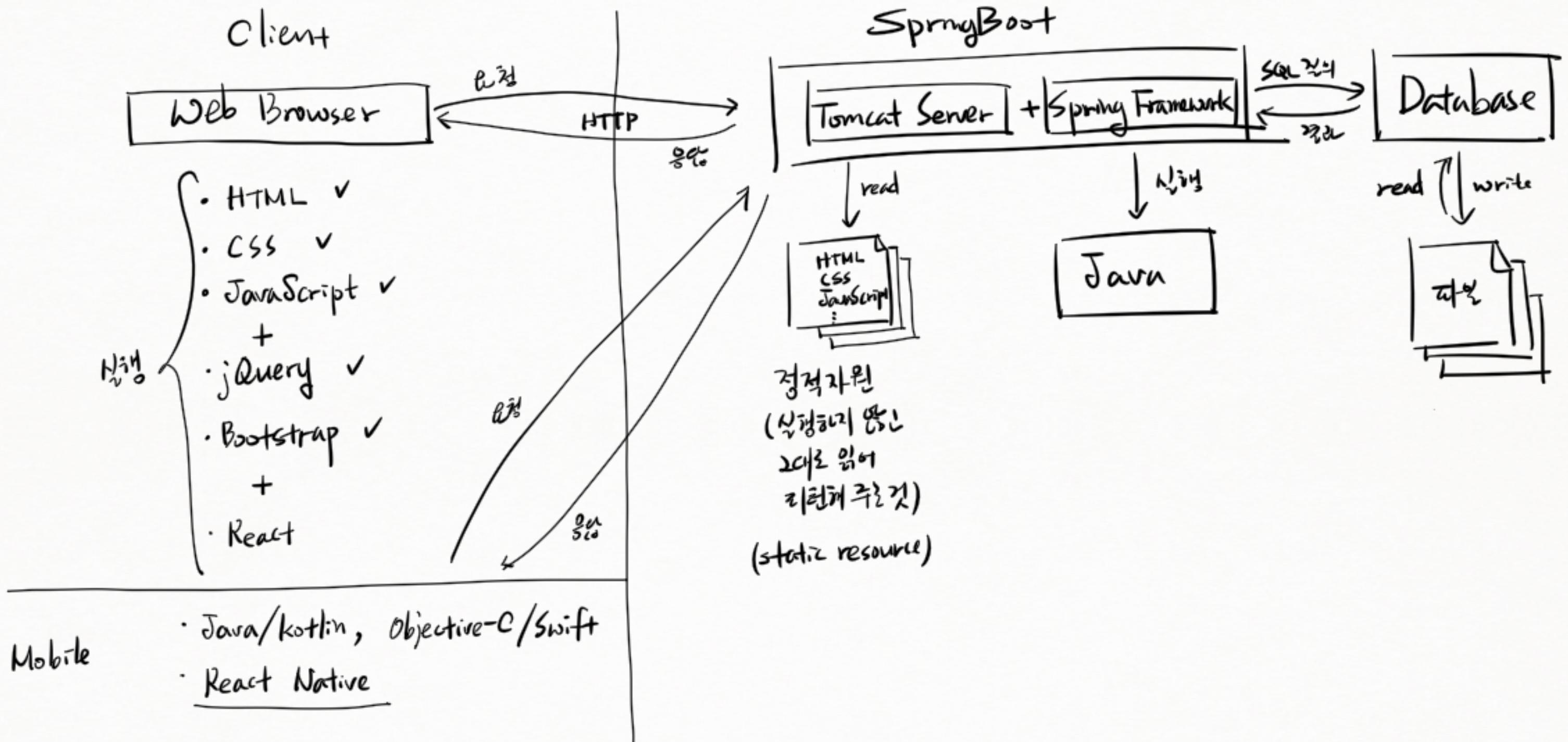
## \* Web Application Architecture by Java



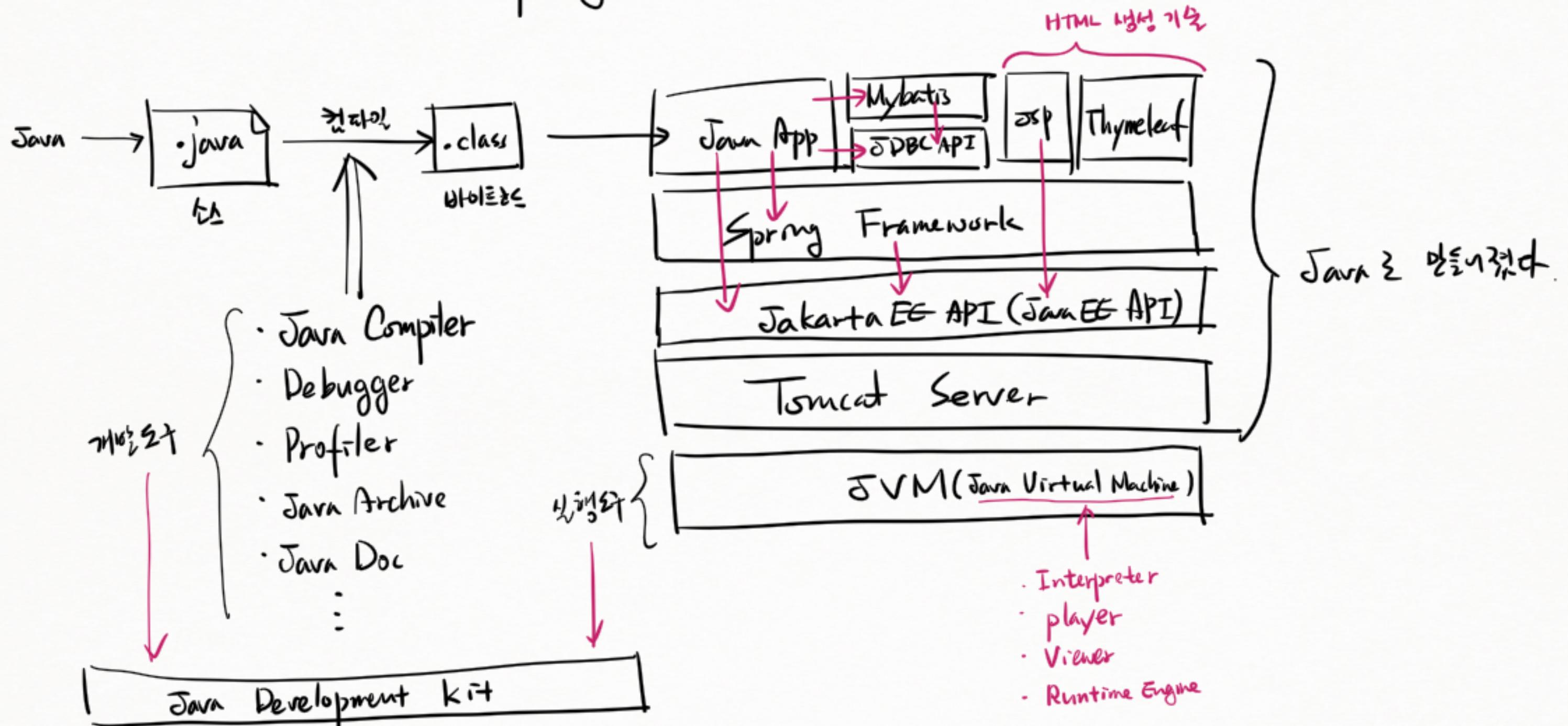
## \* Node.js Web Application Architecture



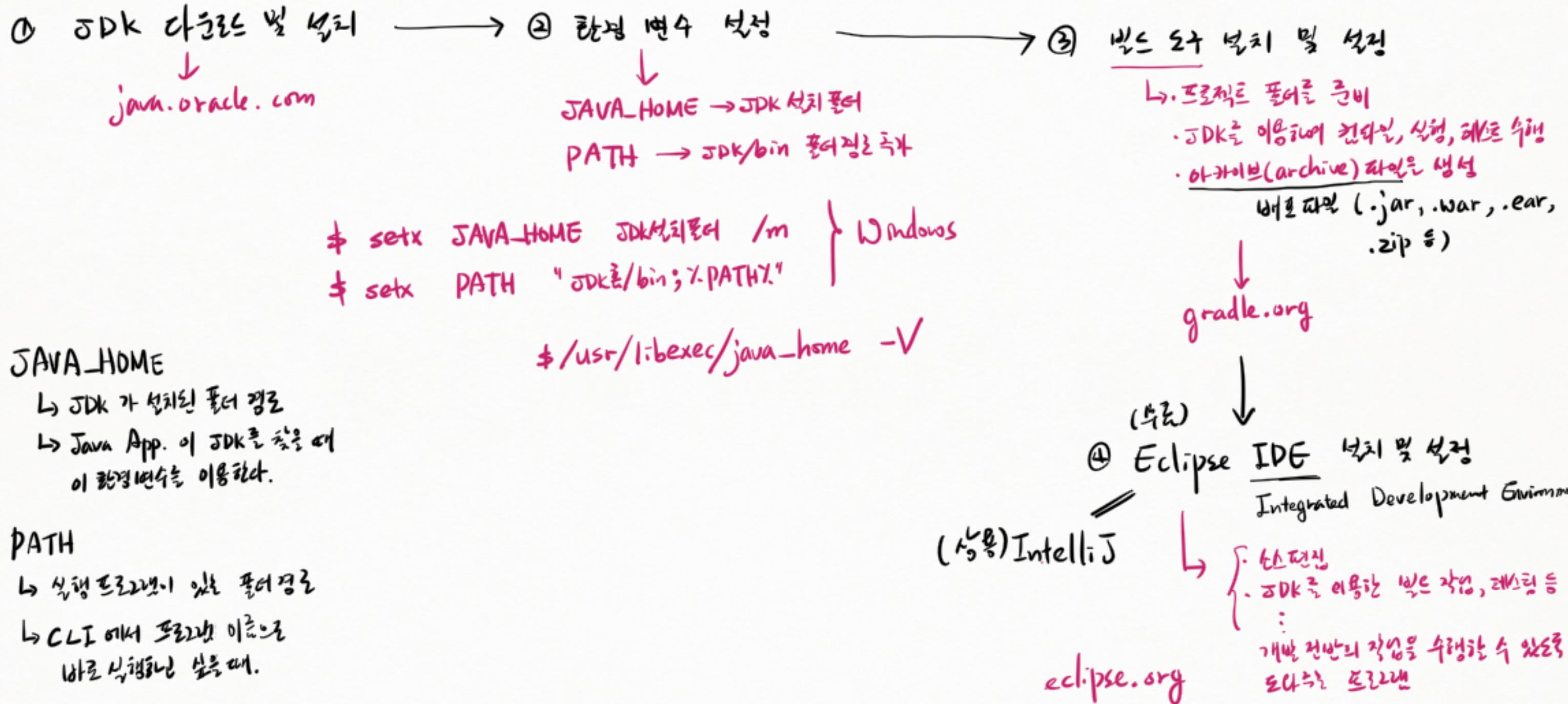
## \* SpringBoot Web Application Architecture



## \* SpringBoot 기술 스택

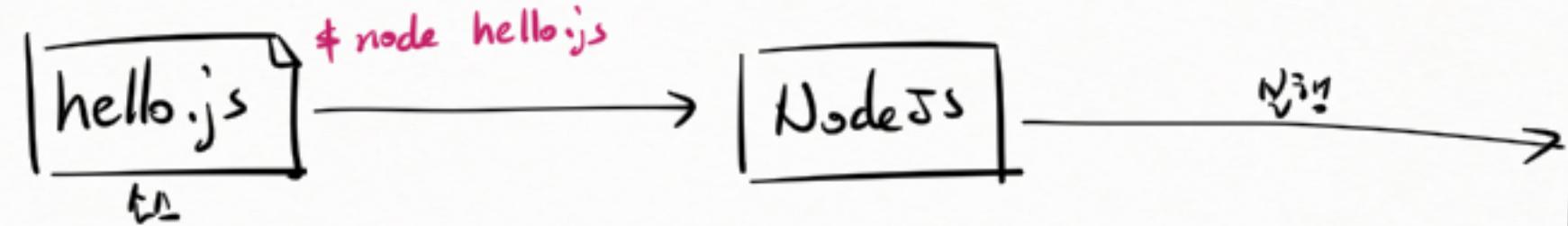


## \* Java 프로그래밍 준비

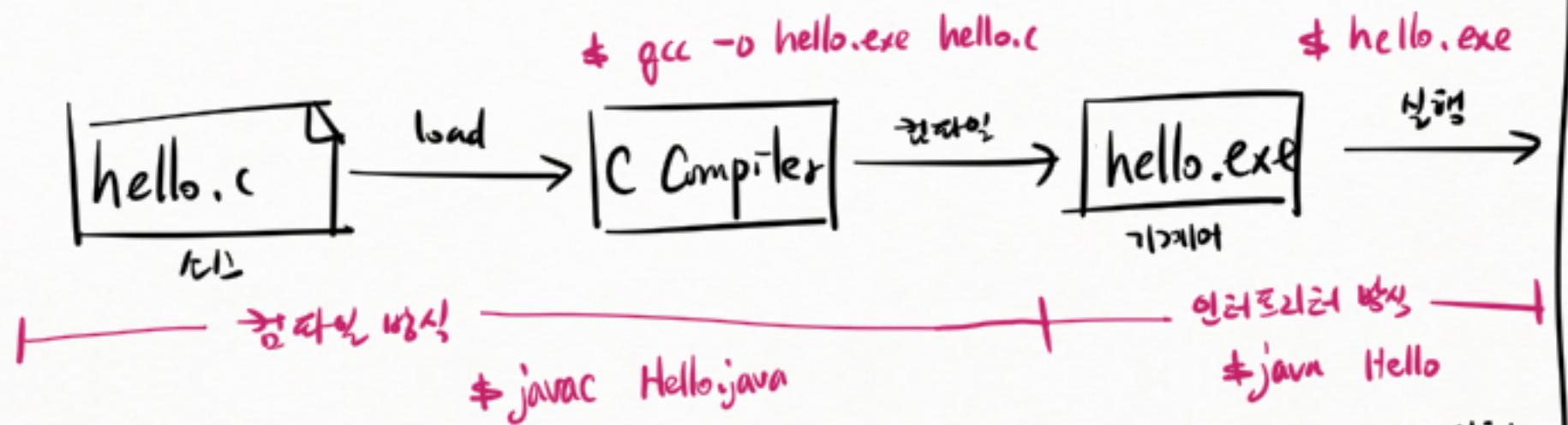


\* App. 구현 방식 및 차이

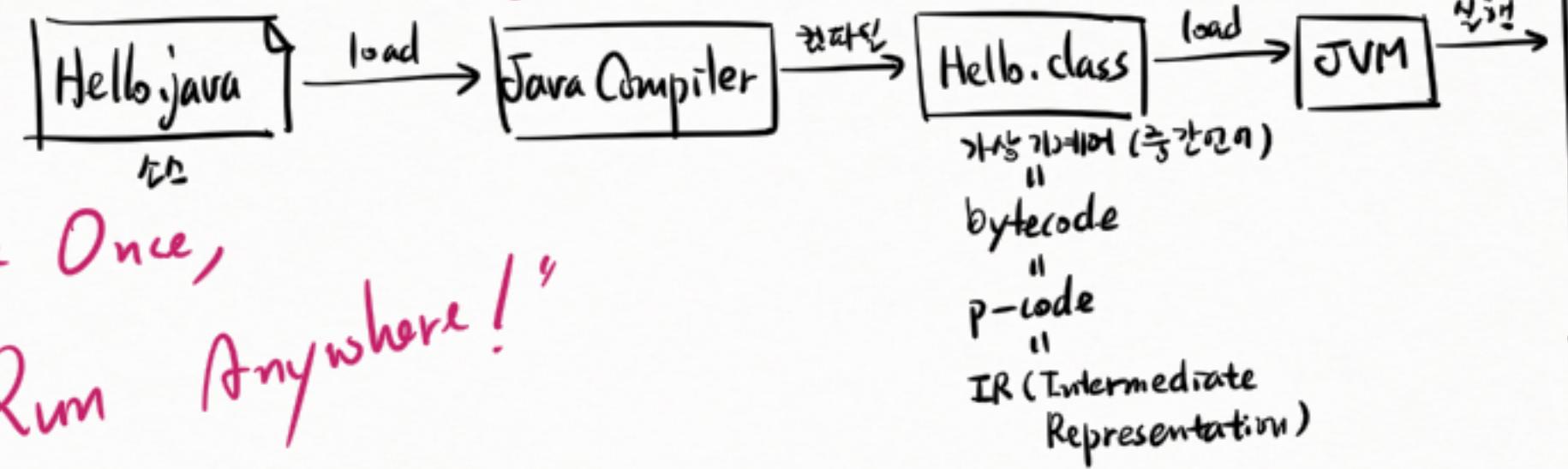
① 인터프리터 방식



② 컴파일 방식



③ 혼합형 방식  
(hybrid)



"Write Once,  
Run Anywhere!"

## \* Hybrid 방식을 도입한 이유

- 선수 간자적인 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 오류검사를

하는  
선수 인터프리터 방식보다  
수행 속도가 빠르다.

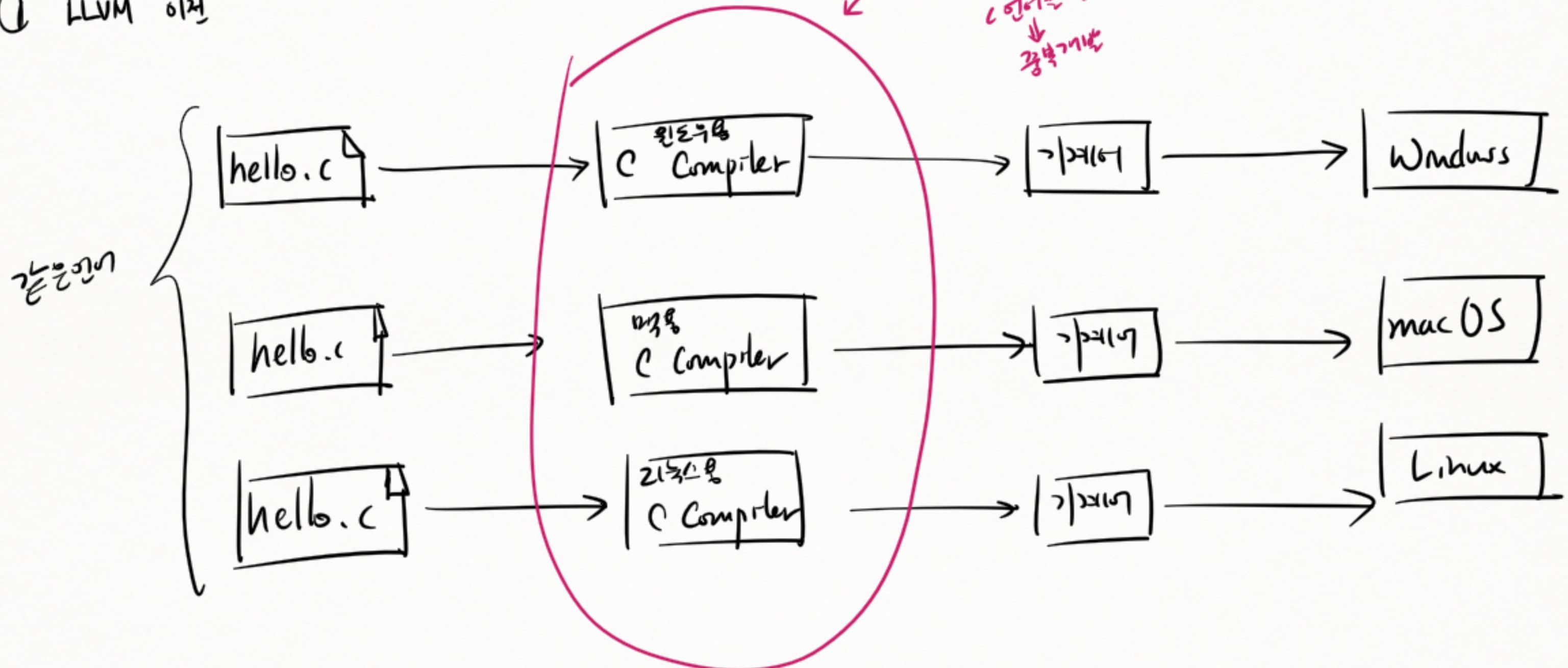
## 선수 인터프리터 방식 보다 나은 이유

- ③ 컴파일과정이 실행의 오류를 모두 찾아낸다
- ④ 일반 기계어는 하드웨어 기계어에 가까운 언어로  
변환된 명령을 실행하니 대용량  
소스에 작성된 명령을 실행하는데 빛난다.  
수행 속도가 빠르다.

\* LLVM 저작자

Low Level Virtual Machine

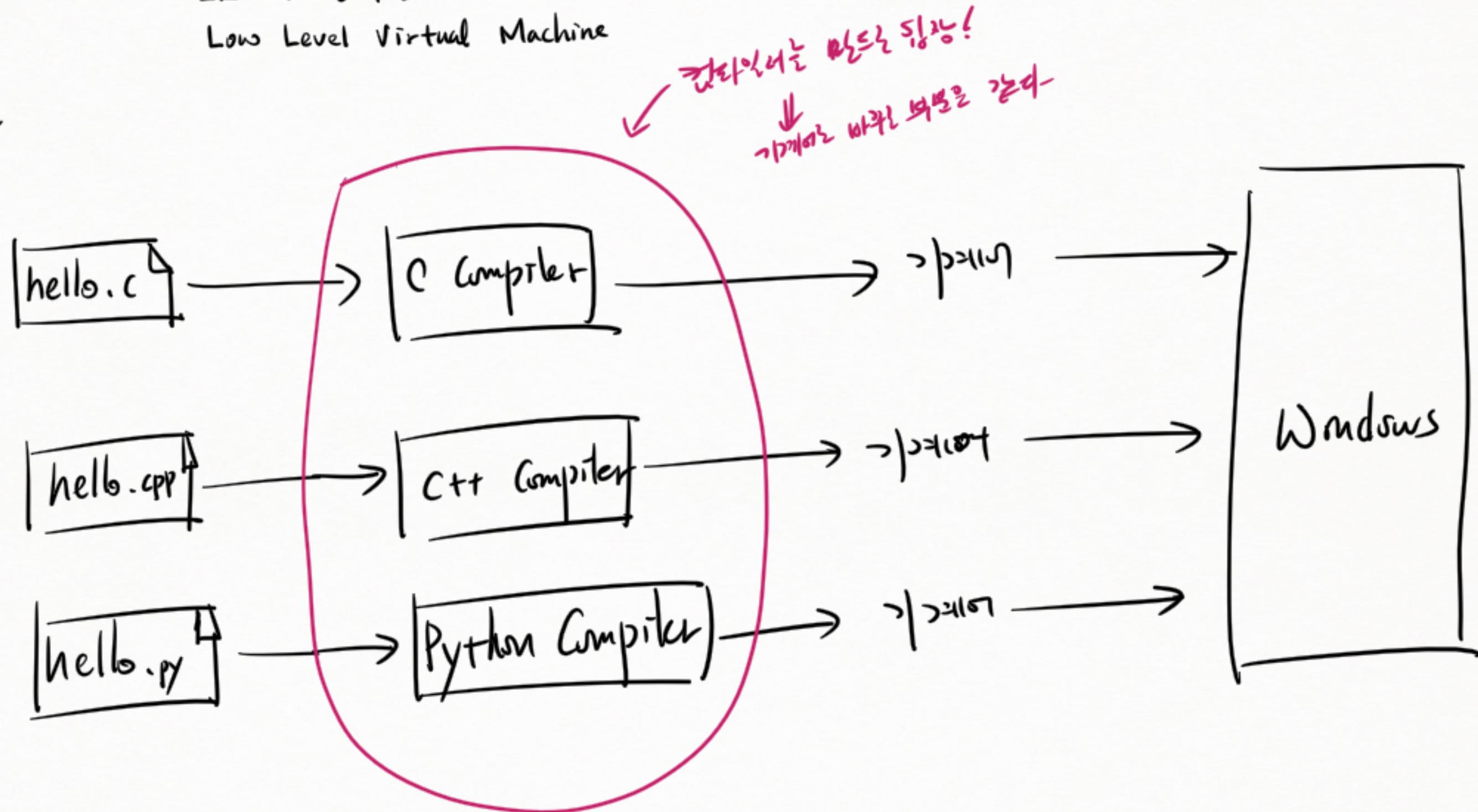
① LLVM 이전



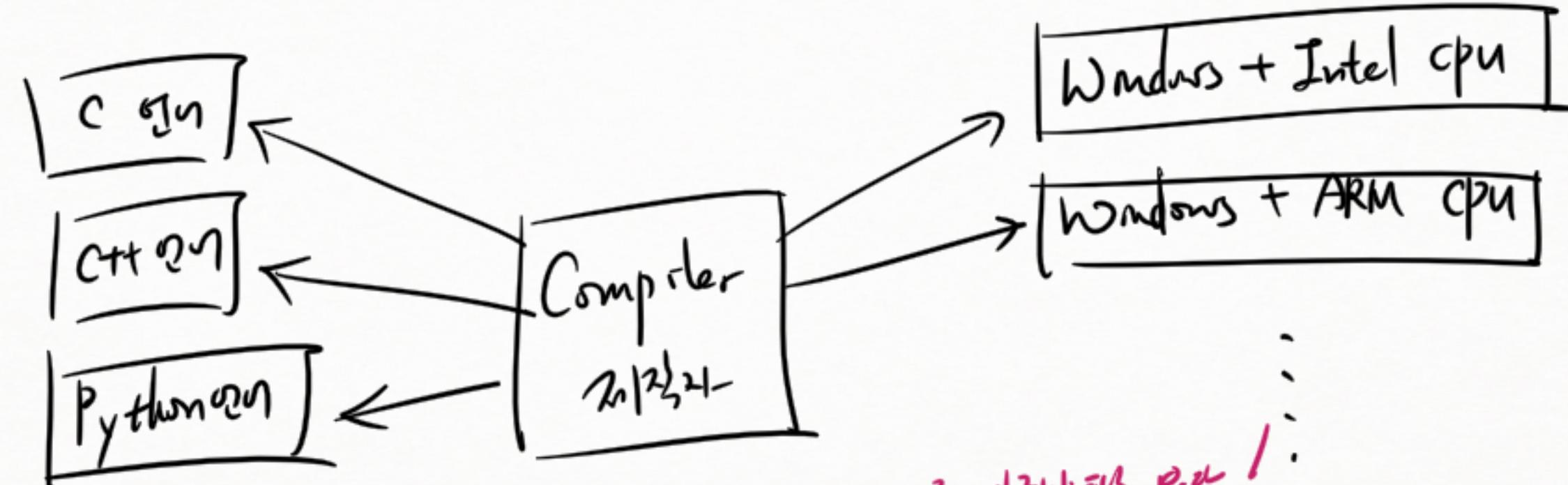
## \* LLVM 커널 파일

## Low Level Virtual Machine

## Q LLVM 이전



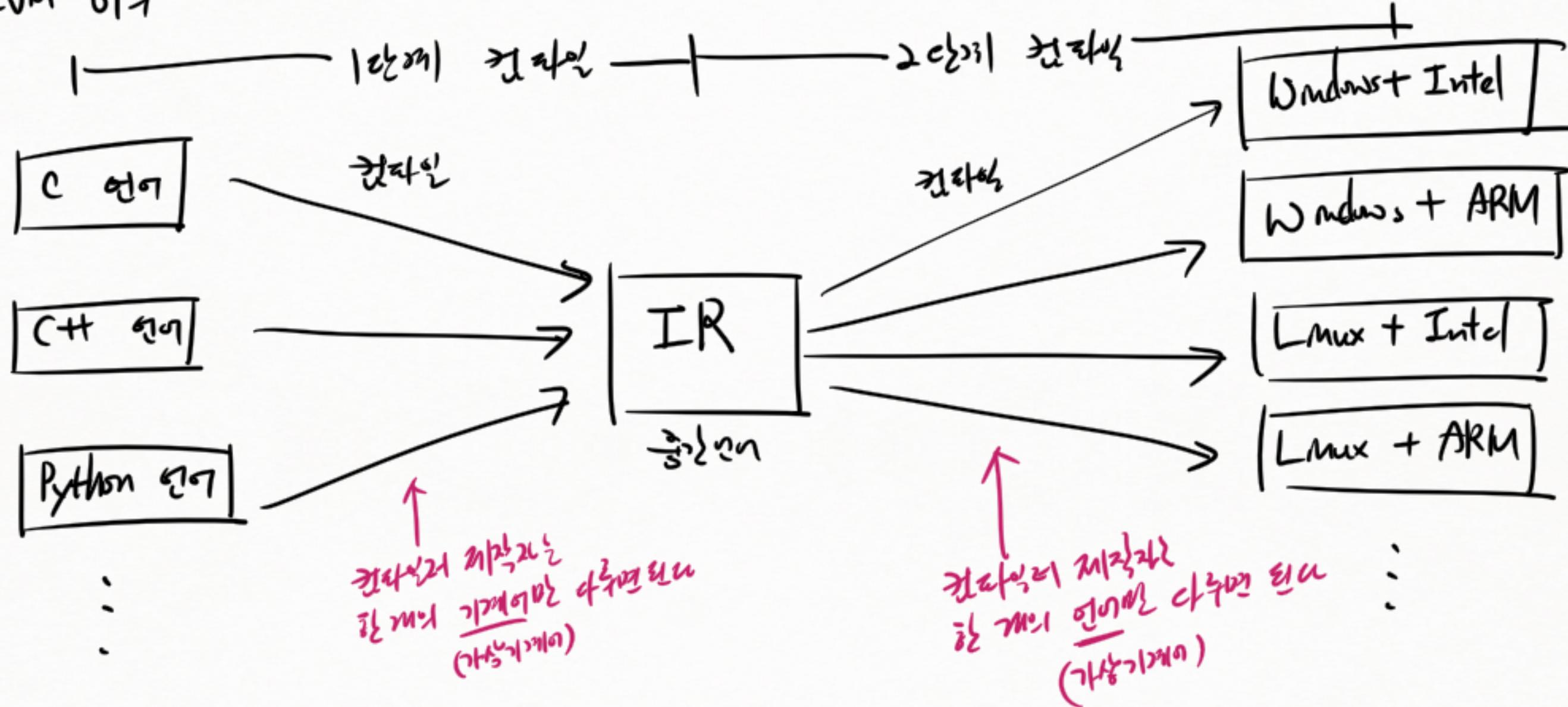
+ 가끔 틀렸을 때 예제



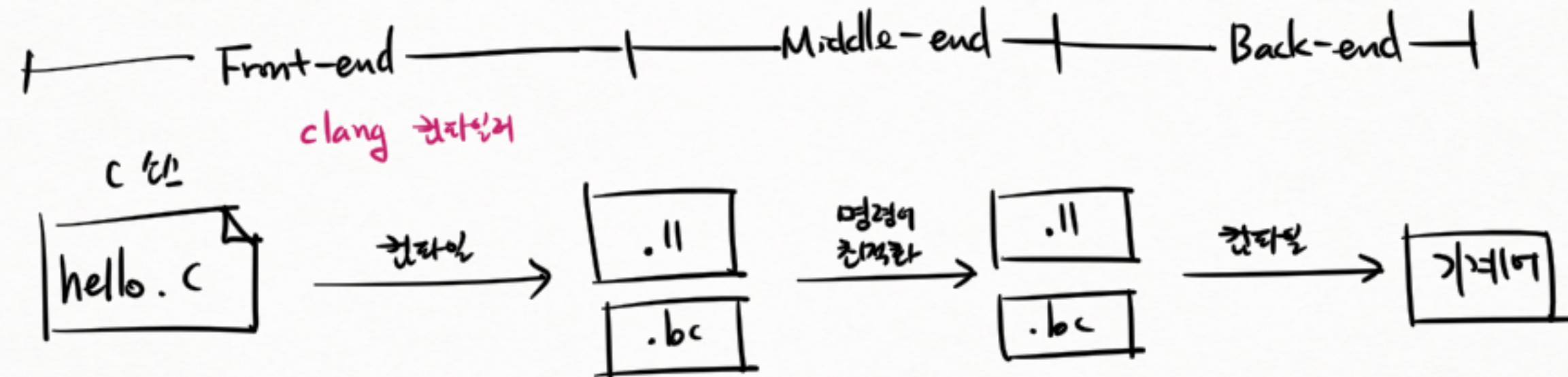
- 어떤 언어를 선택해도 되는지!

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

## ② LLVM 이후



## \* LLVM 컴파일 과정

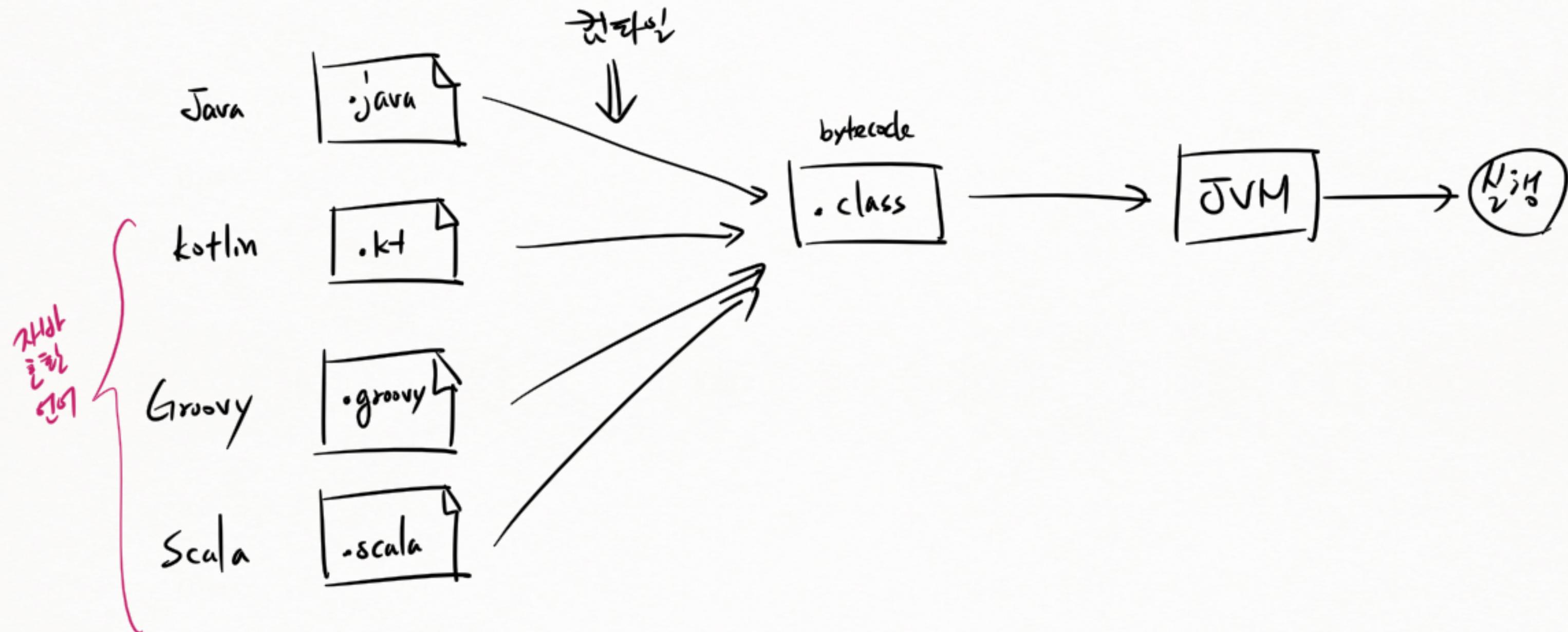


LLVM Assembly

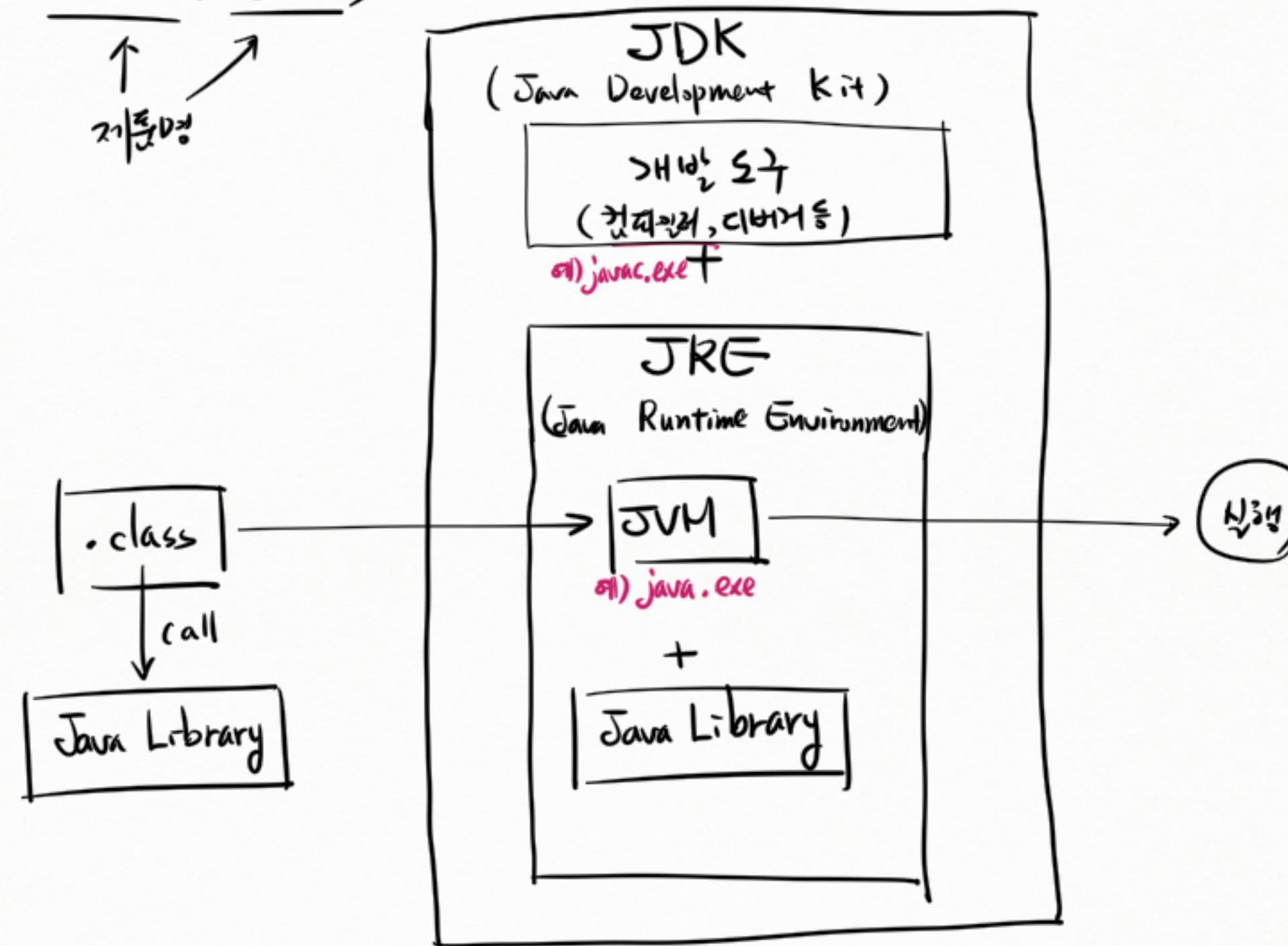
LLVM bitcode

LLVM bytecode

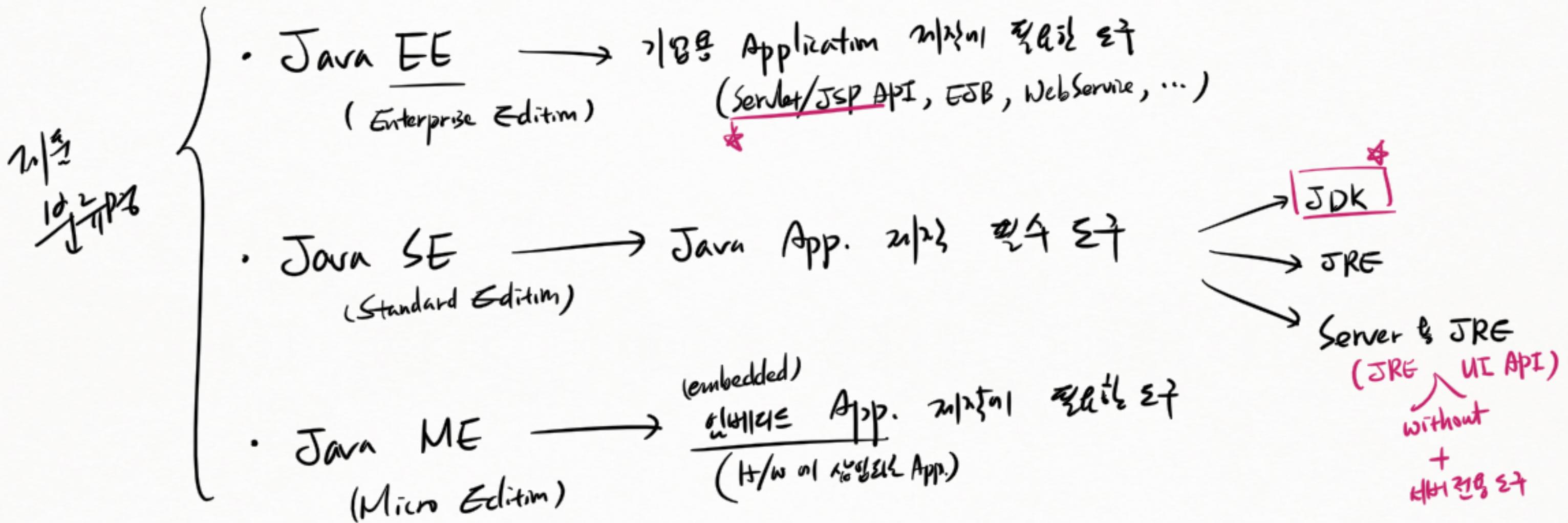
## \* Java et LLVM



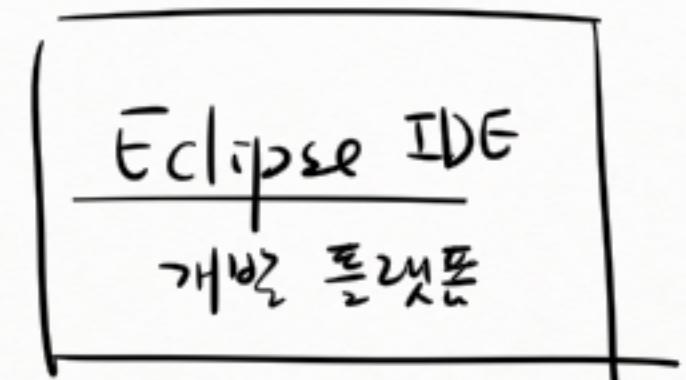
\* JDK, JRE, JVM



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



\* Eclipse IDE



+ plug-in ⇒ 개별 도구 박스

- JDT
- CDT
- :

## \* Java Project 폴더 구조

① 프로젝트 폴더 생성

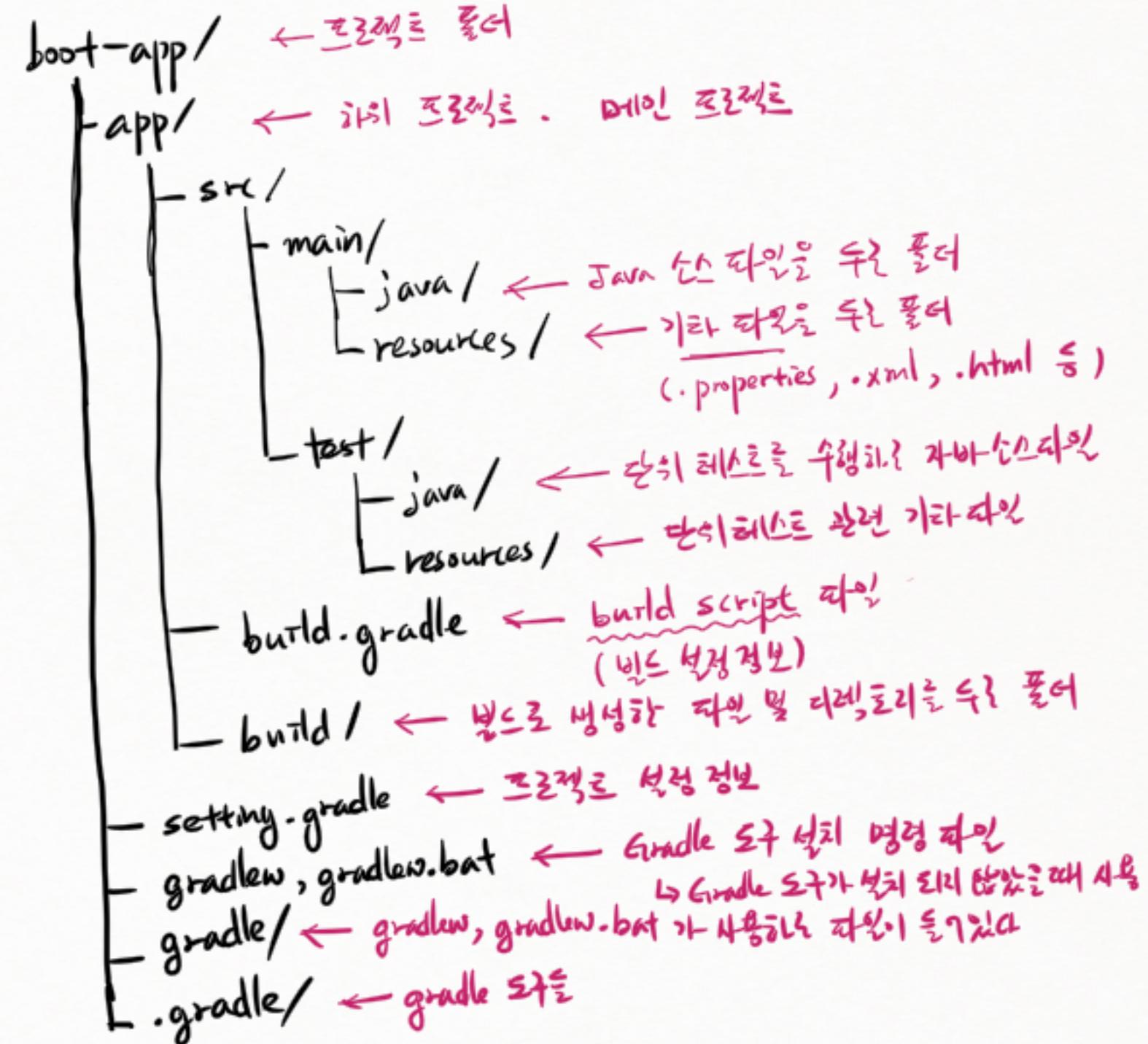
c:\Users\bitcamp\git\bitcamp-ncp\  
User Home  
↳ boot-project\

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

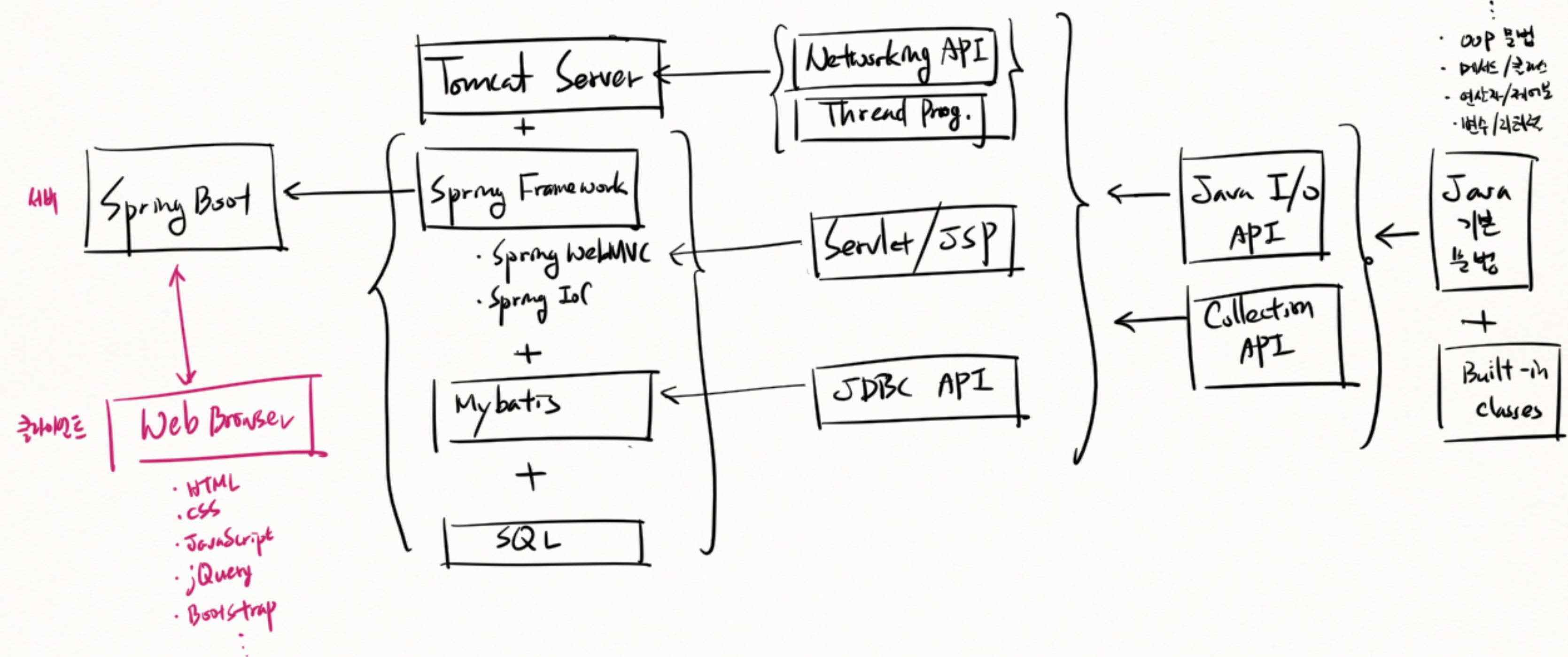
- Dependencies:

- ① Spring Boot Dev Tools ← 자동로딩

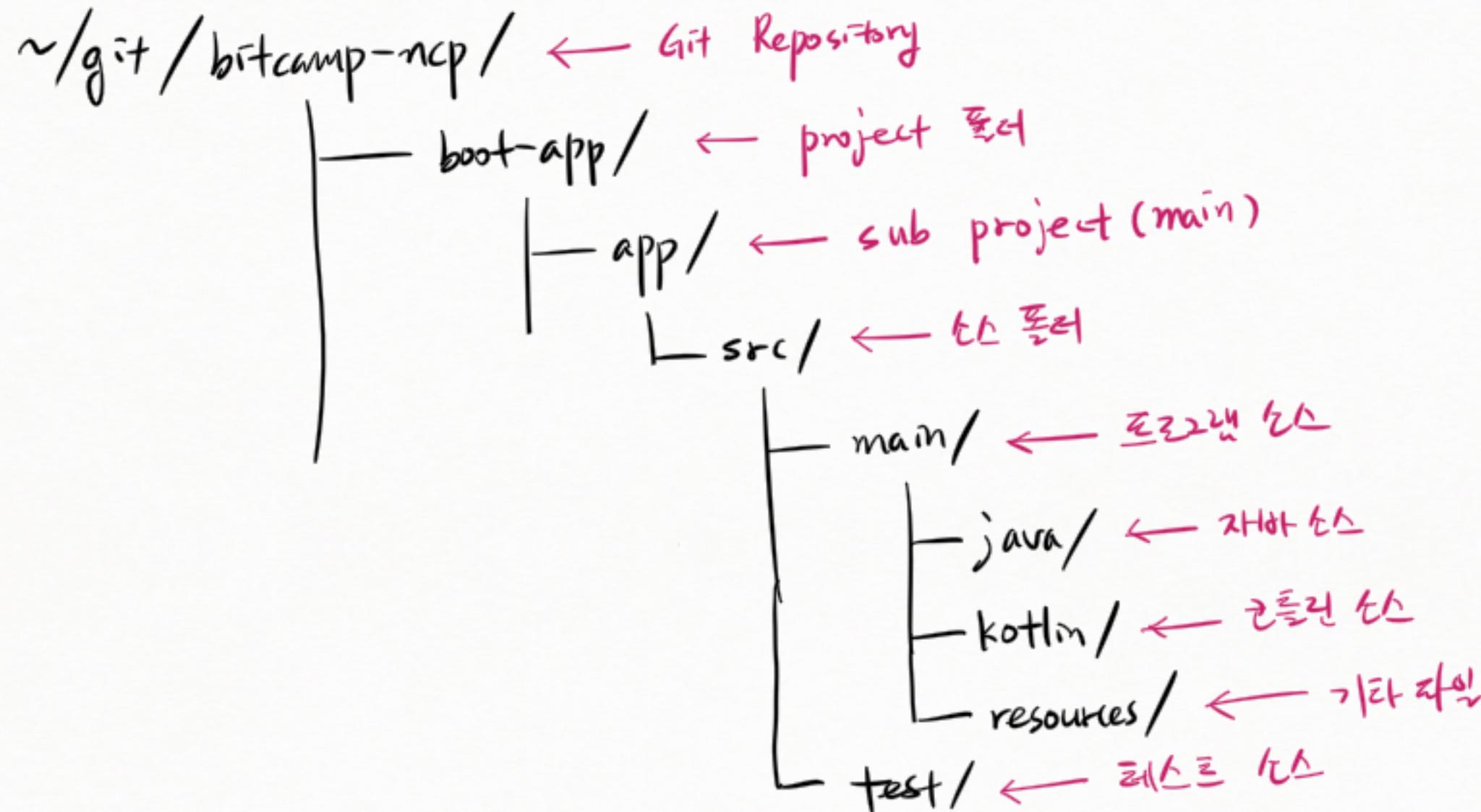
- ② Spring Configuration Processor ← properties 대체  
자동로딩

- ③ Spring Web

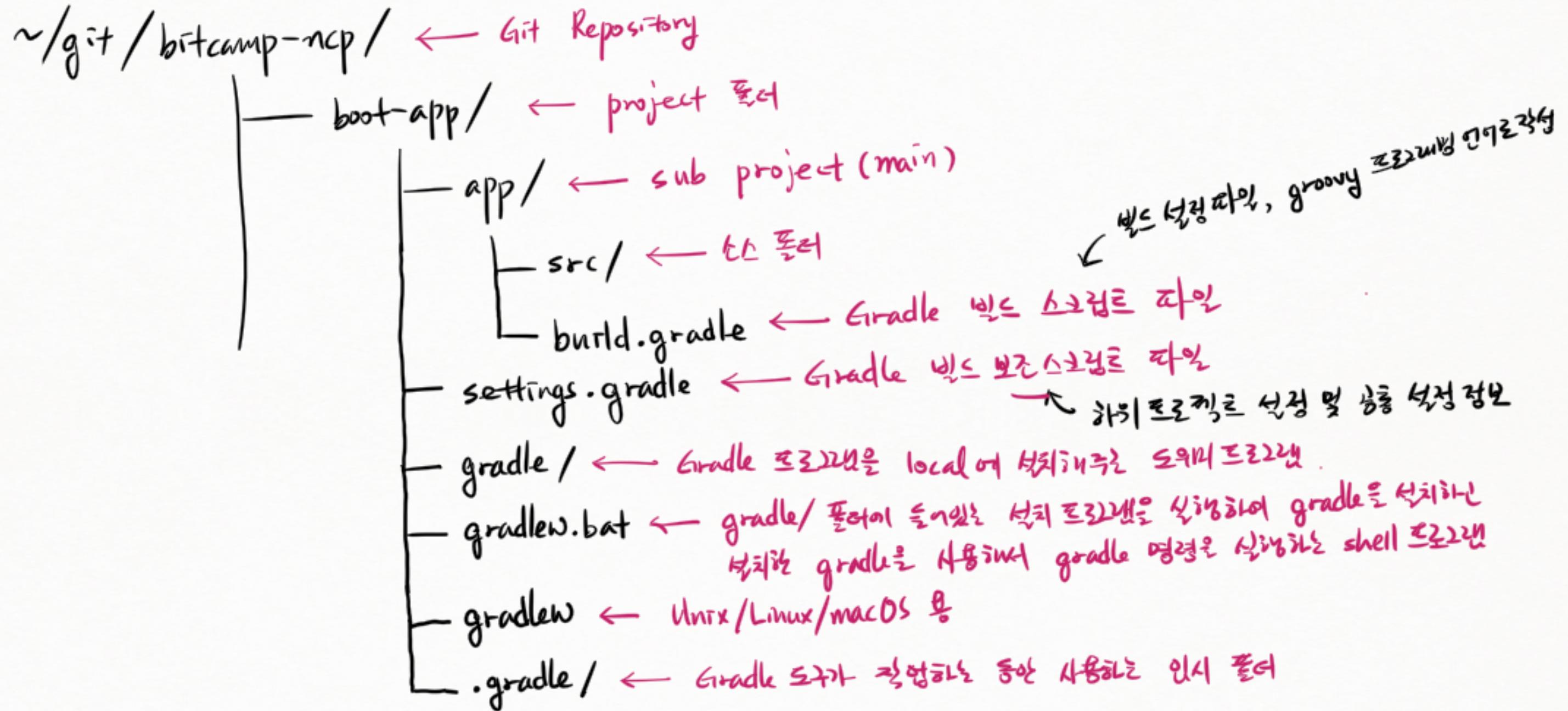
## \* Java 와서 Spring Boot 까지



\* Maven 워크스페이스의 좋은 프로젝트 디렉토리 구조



## \* Maven 워드넷의 풋을 프로젝트 디렉토리 구조

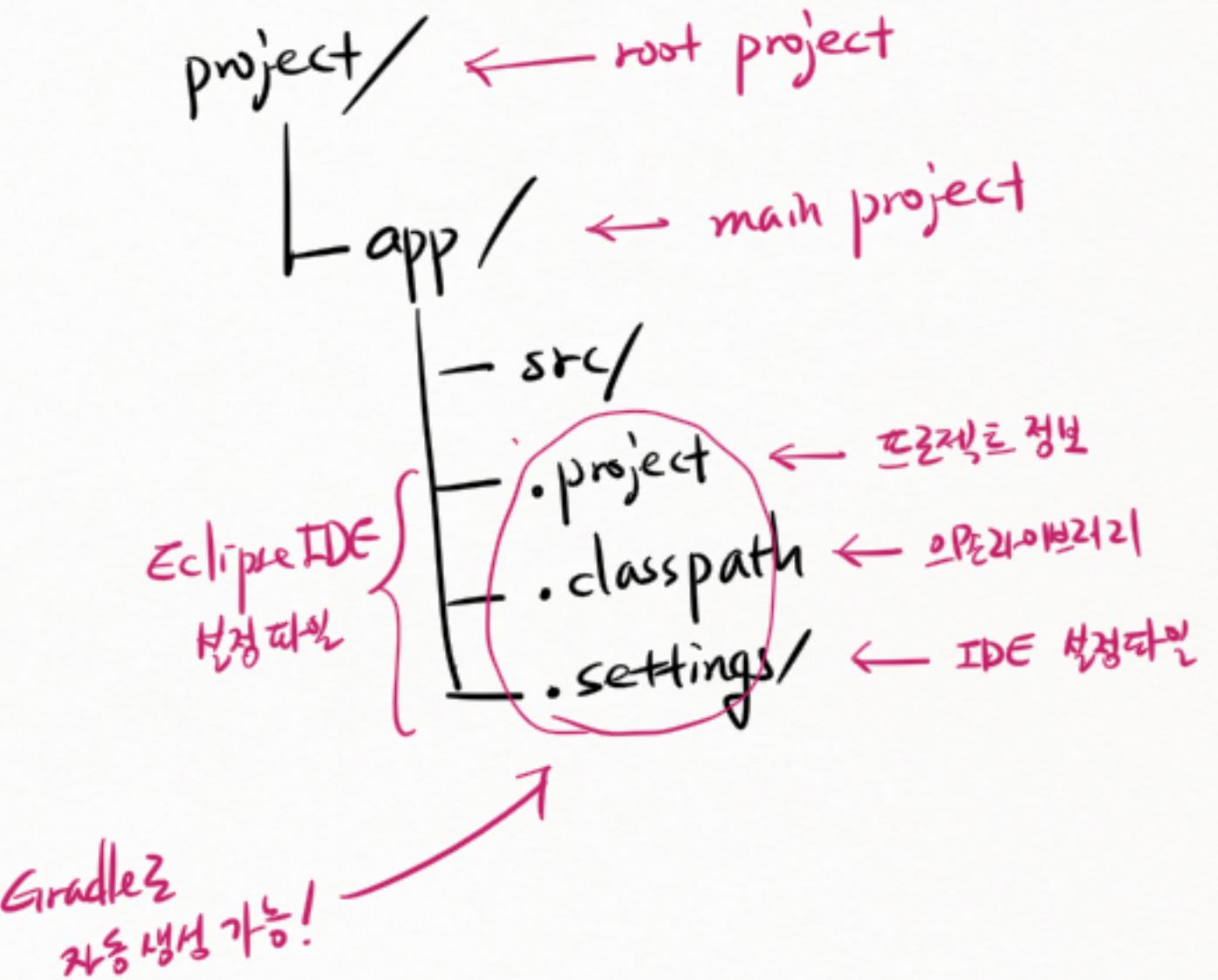


문법 응용 실전 프로젝트 : myapp  
↓  
Back-end 프로젝트 : backend-app  
Front-end 프로젝트 : frontend-app

문법



\* Eclipse IDE 와 프로젝트

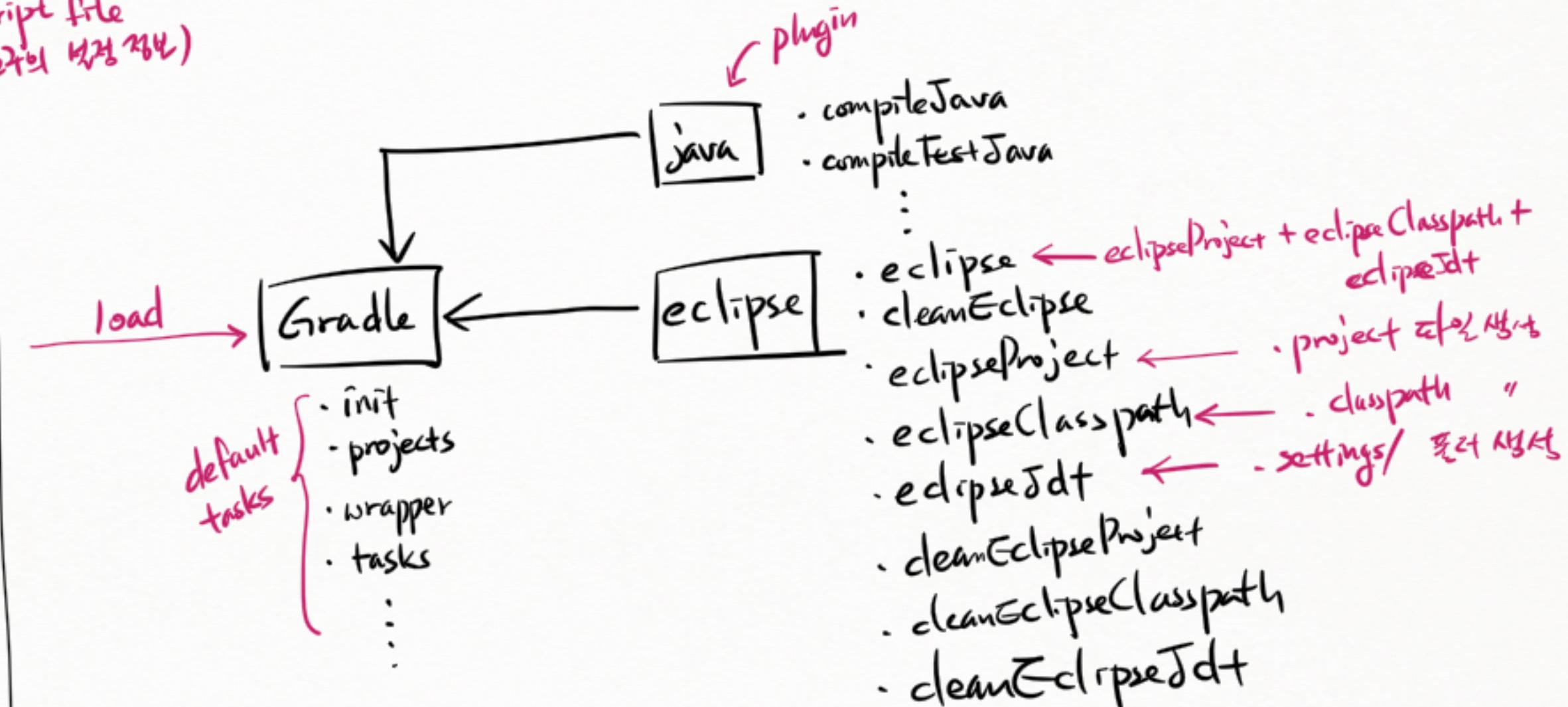


## \* Gradle 빌드 시

[ build.gradle ]

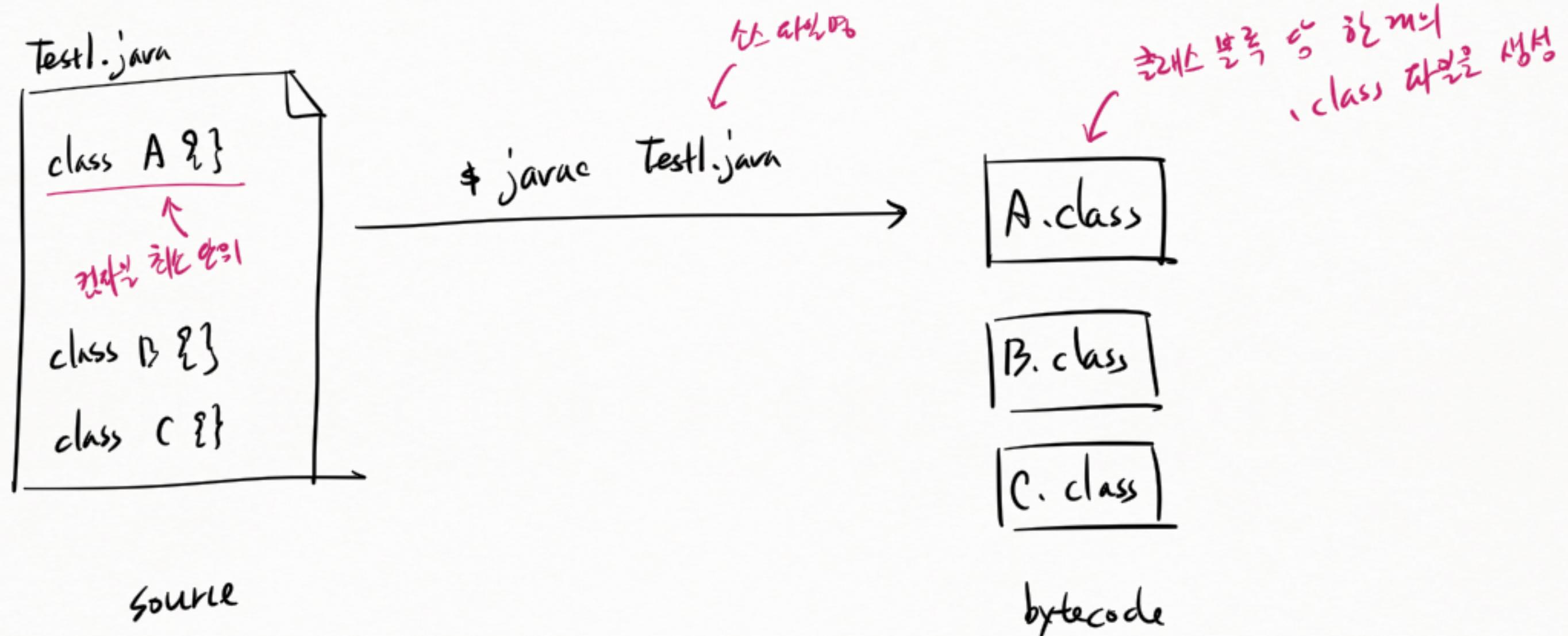
```
plugins {  
    id '풀려고인 ID'  
    :  
}
```

build script file  
(Gradle 설정 파일)

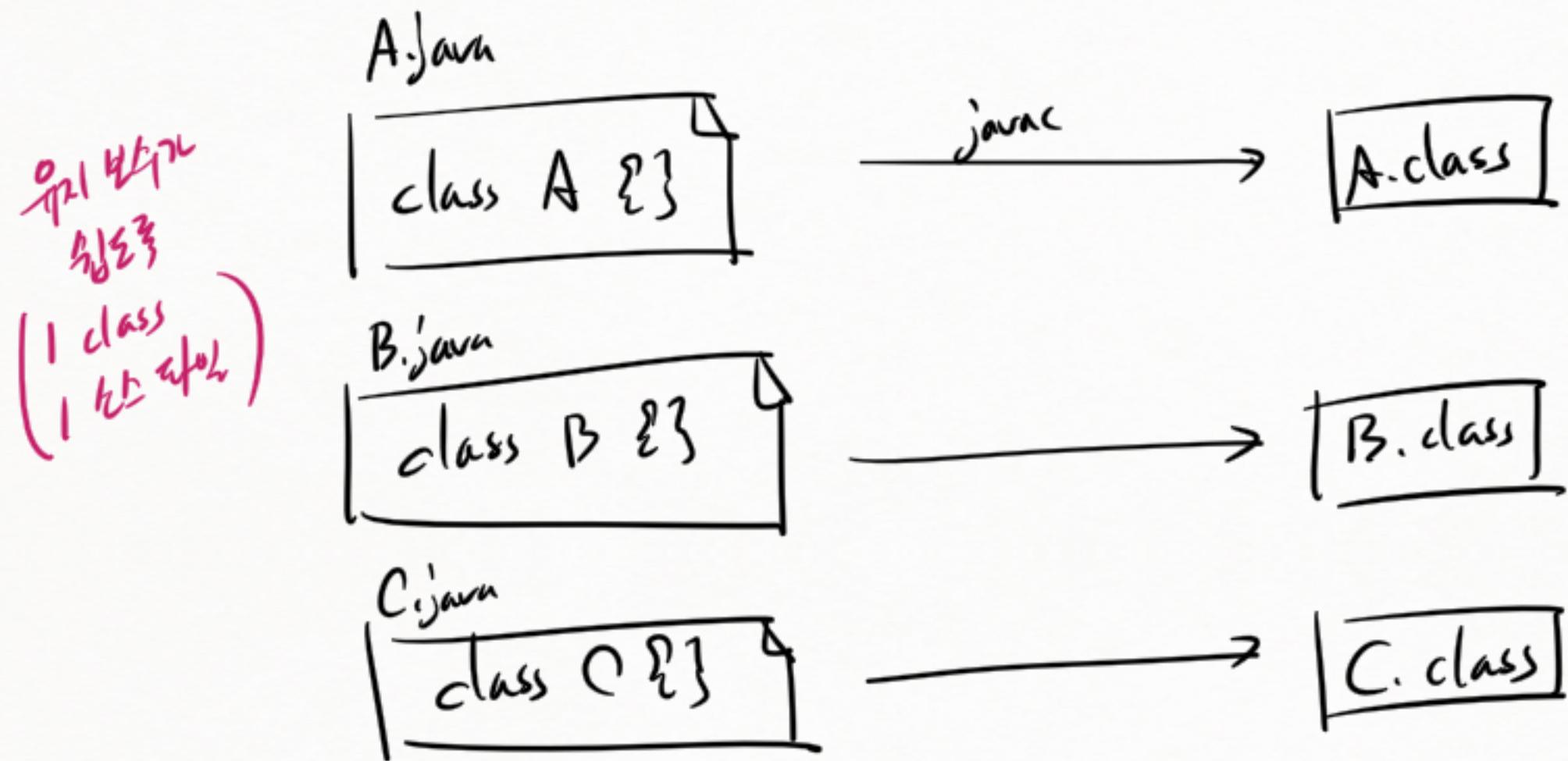


Java ဂျာများ

\* .java 와 .class , → 디자인



\* class မှတ်လုပ်ခဲ့



\* 한 번에 .class 파일 뿐만  
↳ 바로 파일을 확인하는 편이 좋다

~~binary~~

src / A.java  
B.java  
C.java

myapp\$javac -d bin src/A.java

↑  
컴파일 결과물 위치

bin / A.class  
B.class  
C.class

## \* 햄스 파일의 디렉토리

↳ 여러 명의 햄스 파일을 관리하기 편리한 키보드 단축키를 하위 디렉토리에  
설정한다.

src / A.java  
| p1 / B.java  
| p2 / C.java

# javac -d bin src/\*.java ...

bin /  
A.class  
B.class  
C.class

→ 클래스들은 .class 확장자로  
소스 파일의 경로와 동일하게  
설정하지 않는다.

\* 키워드는 아이디어

↳ 코드 블록은 구조화된 실행 블록

src/A.java

```
class A {}
```

src/B.java

```
package p1;  
class B {}
```

src/C.java

```
package p2;  
class C {}
```



\$ javac -d bin src/\*.java

bin/A.class

```
p1/B.class  
p2/C.class
```

\* 터터 위는 아님

src/A.java

```
class A {}
```

src/p1/B.java

```
package p1;  
class B {}
```

src/p2/C.java

```
package p2;  
class C {}
```

터터 위는 찾기 쉽도록  
터터 위에 package 이름  
같은 헤더를 넣어.

# java -d bin

src/\*.java src/p1/\*.java src/p2/\*.java

bin/A.class

  |  
  p1/B.class  
  |  
  p2/C.class

1

\* 대상자와 대상화

```
package pl.px;  
class C {}
```

src/  
|---pl/  
|---px/  
|---C.java

→ 컴파일  
→—————>

bin/  
|---pl/  
|---px/  
|---C.class